ZYBASIC MANUAL

Interak

Document Ref: ZYBASIC-2/1

Issue: 1.1

Date: September 1984

ZYBASIC 2 USER MANUAL

ISSUE 1

(For Versions V.03A and V.03C)
(See Appendix E for Notes on Versions V.04A and V.04C)

Manual Price £3.75

1.2 PREFACE

A Manual usually makes very dry reading; in a way it is a legal document after all, full of rules and regulations. This is fine if you know all about the subject, and only need to consult the Manual infrequently for explanation of particular points, but it's not so fine if you know so little that you have to wade through the whole thing.

A telephone directory is an excellent work of reference, but it makes very boring reading.

An attempt is going to be made in the writing of the Manual to provide for the needs of these two classes of reader, i.e. those who want to read it, and those who don't.

We have found it is very easy to write boring and stodgy material, and so the boring part of the Manual will be no trouble at all. The major difficulty comes in the writing of the interesting part. We want to let the new user know he's not alone; we want to help him or her with the problems which tripped us up when we were learning, and most of all we want the user to have some fun with his computer, regardless of whether or not the task in hand is a serious one.

One way of getting our attitude across is to write as light-heartedly as possible, perhaps including a few jokes. It is a very dangerous technique, because it is very easy to descend to the level of schoolboy humour (or even lower, to undergraduate humour). If the writing is merely facetious it will serve merely to irritate-and-pain rather than entertain-and-explain, so we would be glad of any comments you may care to make on how you view some of the "experimental" material which is contained within.

1.3 CONTENTS

(Total number of pages Issue 1 Manual: 117)

1.4 ERRATA

Issue 1.1 September 1984

2.1 ZYBASIC 2 General Features

* Floating Point Control BASIC Adapted for Interak 1 Computer System - Approximately 6K in Length.

* Easy to Use.

* Low Cost.

* Works with ZYMON 2.

* DTI-1 Compatible - User Programs may be Stored on Tape.

* Normal Version is RAM-Based, and is supplied on 2400 baud ZYMON 2 Compatible Cassette Tape. (Version "A", Located at A000.) This version runs in RAM at 4 MHz with no Wait States. Loads in about Half a Minute. Occupies no Space when not Loaded (Unlike EPROM based Version).

* Optional EPROM Based Version Supplied in 4 off 2K x 8, 5V EPROM. (Version "C", Located at C000.)

* Floating-point Arithmetic from $\pm 1.5 \times 10^{-39}$ to $\pm 1.7 \times 10^{+38}$.

* Hexadecimal Input Option.

* Has Special Commands to suit VDU-K ("Chunky") Pixel Graphics (Pixel SET, RESET, POINT).

* Will also Work (but with Reduced Performance) in Systems Using Early Kemitron VDU-A,B,G.

* A special version for the VDU-2K (VDU-K user-modified for 64 columns) is available from the User Group.

* 260 Numeric Variables.

* Up to 26 String Variables (Full Memory Required), variable maximum length 255 Characters.

* Will Work with Reduced Performance in Systems with Less Than Full RAM.

* Includes Built-in Printer Driver Routines.

* Manual May be Purchased Separately.

2.2 Introduction To This Manual

You are one of two people. You may be one of many more, but for now let's say you're one of two. One of the two people knows all there is to know about BASIC: he's going to be mortally insulted if I try and teach him about line numbers, GOSUBs, GOTOs and so on. If I try he's likely to say GOTO himself: "GOTO @%**!" (@%** being the place you go when you're dead if you haven't paid your BASIC Sin-tax). May I quickly direct him to skip over the beginning of this Manual, and simply consult the long boring lists of commands as and when he feels the need. (Even the chap who knows it all will need to take a look from time to time because, even though I say it myself, this BASIC is "Kinda Cute"; I hope you'll forgive these technical terms at such an early stage. By the way I can say these complimentary things about ZYBASIC 2, because I didn't writa byta it, I'm as not as clever as I sound (!))

By the time we get to this paragraph the chap who knows it all will have taken the hint and skipped all this junk, so that just leaves you (the "other one" of those two people I mentioned), and me. In a way I'm quite glad he's gone, because I can't stand people who know it all. The "other one" knows not a lot about computers, and not a lot about BASIC, and by now is wondering what he's got himself into. If you feel you aren't as dumb as all that clear off and join that know-it-all chap.

. . . I get this funny feeling that I'm on my own now, but I'll press on regardless. It'll pass the time until they send the van with those two nice men with white coats to collect me.

However, I've started; so I'll finish. I hope you don't mind me writing in this silly style, but what I'm hoping is coming across is that I think this whole subject should be FUN. People in computers (no no silly, I don't mean in computers literally! remember the proverb, people who live in computers shouldn't throw stones), well people in computers seem to get very intense about things. They are neurotic about making sure that they got more of whatever it is they want than the next guy (or doll). If he's got a 2K BASIC then I want a 4K BASIC, if he's got a 6502 then I want a Z80, if he runs at 2 MHz, I want to run at 4 MHz, and so on. Perhaps I'm just as neurotic myself because I've got a 4MHz Z80 and I'm wanting to run at 6 MHz - is there any way of working out if it's I who is mad and the rest of the world sane or if it is I who is sane and it's you lot who are mad? (It reminds me of the time I dreamt I was awake and found out I was asleep after all.) Anyway existentialism has no place in a BASIC Manual, so let's get back to the point.

To repeat, fun is what it's all about. I bet the computers are having fun watching us - we all know computers are always taking our jobs away, but have you noticed they always seem to take the best jobs. I mean, when was the last time you heard of a computer volunteering to take a job like unblocking a drain, or washing the car?

Don't misunderstand me; just because we're going to have some fun, we still need a pretty good BASIC, in fact one written with the user in mind. It's similar to the fun some people get out of free-fall aerobatics, they find it's not so much fun without a decent parachute.

2.3 Introduction to ZYBASIC 2

This implementation of the BASIC language is intended as a fundamental step in the growing software pool (mind you don't fall in it) for the Interak 1 computer.

It renders the original 2K Tiny BASIC entirely obsolete, as it has many new commands; the most valuable addition is floating point arithmetic, which makes the computer a real tool in the hands of the programmer.

Although old Tiny BASIC programs can be loaded and run under ZYBASIC 2, it is certain that numerous detailed syntax changes will be needed. The opportunity has been taken in producing ZYBASIC 2 to bring the syntax into line with similar BASIC interpreters for other machines, but this rule has not been followed rigidly: there are commands in this BASIC which haven't been seen anywhere else, and there are places where the ZYBASIC 2 Syntax is better than that of commands for other BASICs (he said modestly).

Release 1 ZYBASIC 2

At release 1 the interpreter occupies a little over 6K of memory. The normal version is RAM-based and supplied on a 2400 baud cassette which loads under control of ZYMON 2. The normal version (Version "A") is located at #A000 (the "#" means hexadecimal numbers, until I tell you otherwise). ZYBASIC loads from the cassette tape supplied in about half a minute.

(As an option a version (Version "C") is available in 4 off 2K EPROMs which are to be located at #C000. This version is not promoted vigorously because the EPROMs cause it to be dearer and slower; also it ruins the idea of Interak being a mainly RAM-based system which gives the ultimate flexibility.)

Both versions are substantially the same and so the majority of this Manual applies to both. Where the difference is significant the fact will be noted.

2.4 Hardware Requirements.

Hardware Requirements for Version "A" (RAM Version)

To install and run ZYBASIC 2A you need:-

1. A Z80-based computer with a working ZYMON 2 Monitor, (including a tape interface of course).

2. 16K of RAM starting at #8000.

3. 16K of RAM starting at #0000.

(A further 16K of RAM starting at #4000 is an optional but worthwhile

addition, since ZYBASIC can use extra memory up to #7FFF for the user's programs.) Users of a single card with 48K or more of RAM will of course already have more than enough RAM.

Hardware Requirements for Version "C" (EPROM Version)

To install and run ZYBASIC 2C you need:-

1. A Z80-based computer with a working ZYMON 2 Monitor.
2. 16K of RAM starting at #0000.
3. Optional RAM for strings, starting at #8000.
4. A RRM-14 card, for the BASIC EPROMs themselves.

(A further 16K of RAM starting at #4000 is an optional but worthwhile addition, since ZYBASIC can use extra space up to #7FFF for the user's programs.) Users of a single card with 48K or more of RAM will of course already have all the RAM needed.

Additional Remarks on Hardware (Versions "A" and "C")

Typical Memory Map (Addresses in hexadecimal notation)

0000 - 0FFF	Immaterial, (Usually Machine-Code Monitor in RAM).
1000 - 7FFF	RAM for ZYBASIC's scratch-pad etc., numeric variable store, User's BASIC source program.
8000 - 9AFF	RAM for ZYBASIC's string variable store (optional).
9B00 - 9FFF	Immaterial (Usually spare RAM).
A000 - BFFF	Version A: ZYBASIC, followed by spare RAM. Version C: Immaterial, (Usually spare RAM).
C000 - D7FF	Version A: Immaterial (Often RAM or EPROM). Version C: ZYBASIC EPROMs.
D800 - DFFF	Immaterial (Usually RAM or EPROM).
E000 - EFFF	CPU Card on-board firmware (e.g. monitor or boot).
F000 - FFFF	Memory Mapped VDU Area + space for VDU-related expansion, e.g. larger format, or programmable character generator).

Typical I/O Port Map (Port Numbers in hexadecimal notation)

More Interak 1 Port Number Allocations are given in the Interak 1 System Notes (the descriptive leaflet supplied free of charge); only the Ports which are directly relevant to the use of ZYBASIC are given here.

(continued on next page)

Typical I/O Port Map (continued)

Tape Cassette:

04 Read: Tape UART Status Port.
Data Bits 0 - 5: Immaterial.
Data Bit 6: DAV (Data Available).
Data Bit 7: TBMT (Transmit Buffer Empty).

04 Write: Tape Card Control Port.
Data Bit 0: Motor on - Cassette 0.
Data Bit 1: Motor on - Cassette 1.
Data Bits 2 - 7: Immaterial.

05 Read/Write: Tape UART Data Port.
Data Bits 0 - 7: Data from Tape / Data to Tape.

Printer:

Although ZYBASIC 2 assumes the printer ("Printer 2") is connected to a serial UART a parallel printer can be used equally well if the printer is connected in such a manner as to require the same control as if it were serial (ask your supplier for details of how this can be done if you have a parallel printer).

06 Read: Printer UART Status Port, or Parallel Printer Status Port.
Data Bits 0 - 6: Immaterial.
Data Bit 7: TBMT (Transmit Buffer Empty), or, NBUSY (Parallel Printer Not Busy)

06 Write: Not Used.

07 Read: Not Used.

07 Write: Printer Data Port.
Data Bits 0 - 7: Data for Printer.

Parallel ASCII Keyboard:

40 Read:
Data Bits 0 - 6: ASCII Data from User's Keyboard.
Data Bit 7: Keyboard Data Strobe.

40 Write: Not Used.

2.5 Installation.

Installation of Version "A" (Cassette Version)

No specific installation is required since ZYBASIC 2A loads from tape like any other program, using the ZYMON 2 "L" command.

Installation of Version "C" (EPROM Version)

Fit the 4 EPROMs into sockets having the following addresses:-

"C000" EPROM into socket addressed #C000-#C7FF
"C800" EPROM into socket addressed #C800-#CFFF
"D000" EPROM into socket addressed #D000-#D7FF
"D800" EPROM into socket addressed #D800-#DFFF

Take the normal anti-static precautions during this operation. (These are detailed in the User Manual for the card in use.) Also ensure the correct orientation of the EPROMs. Pin 1 is indicated by markings or indentations at one end of the EPROM encapsulation.

Power On / Cold Start (Version "A" and Version "C")

When the computer is switched on ZYMON 2 will be in control:-

```
 ZYMON 2.VXXX
 ENTER COMMAND
>_                                   ("_" represents the cursor blob)
```

If ZYBASIC 2A is to be used it will have to be loaded from tape:

1. Select the 2400 baud rate on the tape interface and, if motor control is fitted, manually turn the motor control relay on.

2. Type "L", "space", "carriage return", and play the tape.

3. After the program has loaded, switch off the tape machine.

(If ZYBASIC 2C is to be used the above procedure is not required since ZYBASIC 2C is already resident at power on.)

Next, type "E", "space", "A000", for version "A", (or "E", "space", "C000", for version "C").

Press carriage return (CR), and ZYBASIC will initialise:-

```
ZYBASIC 2.XXX              XXXXX
New file opened
>_.............................
```

The above output will be at the top of a blank screen. There will be a variable number at the top right-hand corner of the screen which shows how much memory is available for your program. (In the versions of ZYBASIC current when this manual was written the figure was 26830 with 48K of RAM, and 10446 with 16K (i.e. 16834 bytes) of this removed.)

2.6 Operating Modes

There are two operating modes: Command Mode, and Execute Mode.

In the Command Mode the computer acts on commands as soon as they are entered; another name for Command Mode is Immediate Mode, perhaps for this very reason. Whenever the BASIC is in Command Mode it displays the prompt ">_" on the screen.

The Execute Mode is usually initiated by typing RUN then carriage return; this starts execution of a program which has been written and is already loaded into the computer. Execute Mode can also be initiated by using a GOTO command followed by a line number.

There are various keys which have special functions, and these are listed later. Also listed are all the other things you need to know, such as the Data Types available, the Operators, the Commands, and so on. If you already know something about BASIC you can go and look for them now, and leave us in the kindergarten to learn all about it at our own pace.

(Good, now he or she (or it) has gone we can have some fun.)

Direct or Command Mode

Commands and functions may be used directly, that is, without a line number and without storing them in the text area.

When you are in Command Mode, you can perform calculations directly (it's sometimes called Calculator Mode as well, I wonder why), or enter programs into the machine (machine being another name for the computer) for later execution.

Try entering DOFF, this being the command to turn off the dots. The way you "enter" something into the computer is to type-what-I-tell-you, followed by carriage return; in this case you type DOFF then carriage return. Assuming you've been working with ZYMON 2, you will know what I mean. Right, off you go, do it!

To turn the dots back on the command is DON, try it. If you are wondering what the dots are for the explanation is somewhere else.

At all times the computer will try and obey your commands, but if it can't understand them it will assume that you have made an error. (This typical of a new and not altogether pleasing attitude the machine has to its betters: I can't understand you, so it must be your fault.)

Try entering a friendly word like HELLO, and see what the computer says - unfriendly little machine isn't it? Try entering YOU ARE ONLY A MACHINE, and see what it says.

You are probably getting the idea of how it operates now, so play fair and try a command it can understand: CLS, which should clear the screen to let you start again.

2.7 Special Function Keys

Carriage Return (CR), also CTRL-M (ASCII Code #0D)

The computer "looks at" the line just typed and takes action on what it "sees". If the line begins with a line number (i.e. some combination of the digits 0 to 9), the computer will store the line in its correct numerical order in the text memory for later execution. Even if the whole program entered so far has to be moved up to make space for the new line it is done there and then. If the line entered does not begin with a line number, the computer will attempt to interpret and execute immediately the statements contained on the line.

Backspace (BS), also CTRL-H (ASCII Code #05)

This moves the cursor back one space on the VDU screen, destroying the character on which it lands. It is used mainly for correcting typing errors if they are noticed quickly enough.

Delete (DEL), (ASCII Code #7F)

This has the same effect as backspace. Two keys have been allocated to this one function so that the user to adopt the one he or she prefers (since some keyboards only have either BS or DEL, not both).

Line Feed (LF), also CTRL-J (ASCII Code #0A)

This key causes the cursor to move down to the next line, or perhaps the whole display to move up one line, which is tantamount to being the same thing.

End of Text (ETX), also CTRL-C (ASCII Code #03)

If you are using the AUTO command then CTRL-C will exit from AUTO. The CTRL-C key is also used to return to Command Mode from an Execute Mode data request such as an input statement.

Form Feed (FF), also CTRL-L (ASCII Code #0C)

This is used only in Command Mode, to clear the screen.

CTRL-N, (ASCII Code #0E)

Get Next Character. (Used when editing, see the ED command description for details).

CTRL-R, (ASCII Code #12)

Get Rest of line. (Used when editing, see the ED command description for details).

CTRL-S, (ASCII Code #13)

Suspend execution. This suspends execution of a program or listing at the first available opportunity; once the command is accepted by ZYBASIC, a letter "P" (for "paused", maybe one day it will say "S" for "suspended") is printed at the top right-hand corner of the screen. If CTRL-C is then pressed a break will occur and Command Mode will be re-entered; pressing any other key will let the execution or listing continue. Sometimes the CTRL-S feature is described as "keyboard interrupt", although it does not in reality generate a genuine interrupt to the microprocessor (nor wreak the havoc interrupts can cause.)

2.8 Statements and Lines

The statements from which the user's program is comprised are entered one or more to a line. Each line must have an integer line number between 1 and 32767. Up to 255 characters can be used for the statement(s) on each line, although such a line will appear as several lines on the VDU, and would need a very wide printer to print out. If a line contains more than one statement the statements should be separated by a colon ":".

A line must be typed complete with its line number (except when the AUTO command is in use, when the line number will have been supplied automatically) and is terminated by (CR).

To delete a line, type the line number followed by (CR). To correct errors in a previously typed line use the ED command (q.v.). When two lines are typed using the same line number only the last one will be retained; this is another way to correct errors.

2.9 Execution

A program is executed by typing RUN(CR). All variables are cleared to zero when RUN is typed. Statement lines are executed in ascending numerical order unless the commands GOTO, ON GOTO, GOSUB, or ON GOSUB, are used to alter the sequence. GOTO "n" (CR) may be used to start execution from a suitable line number "n". The line number "n" can be the first line in a program if it is wished to start a program without having all the existing variables cleared to zero first.

2.10 Abbreviated Command and Keyword Entry

Some commands may be used in abbreviated form. When applicable this short form is shown in the command description.

2.11 Line Buffer

This is a reserved space in the system memory in which complete lines ("statements") of a program are stored temporarily while they are being entered, or later for editing. There is space to contain 254 characters. A single line can comprise several shorter statements provided they are separated by the colon (":") character.

2.12 Pause/Continue

CTRL-S to pause; then CTRL-C to HALT, or ANY Key to continue.

2.13 Hardcopy (Printout)

In direct mode PRON and PROFF turn the printer output on and off respectively. The commands PRON and PROFF may be used in direct mode or in a program statement.

2.14 Error Messages

There are three types of error message. HOW, ERROR and SORRY. The offending line of a program is printed with a question mark, indicating where in the line the problem occured.

HOW indicates that ZYBASIC understands the statement but cannot execute it; e.g. division by zero, or a GOTO to a non-existent line number, or a calculation the result of which exceeds the arithmetic range of the ZYBASIC, or a SET, RESET, POINT command that goes out of range.

ERROR indicates that the ZYBASIC cannot understand a command; e.g. an abbreviated command without a dot, a mis-spelt command, one that does not exist, an unmatched bracket, or a missing colon between statements on one line, or say a "NEXT" command encountered before the "FOR" which should have been executed first.

SORRY indicates insufficient memory and may occur while typing in or loading a program, or during the execution of an array statement which tries to use an array location that cannot exist (due either to insufficient memory or the use of an invalid array index value).

2.15 Scientific Notation

ZYBASIC uses scientific notation for the output of very large or very small numbers. This is simply for convenience of printing as a number such as 2,432,900,000,000,000,000 takes up a lot of space when written out in full: in scientific notation it is written as 2.4329E18, where E18 shows that the preceding number should be multiplied by ten 18 times. ie. the decimal point should be moved 18 places to the right. The E stands for exponent and the number following the E can be positive or negative. If it is negative it indicates that the number is to be

divided by ten a number of times. (shift the decimal point left) e.g. 1.4E-3 represents 0.0014. Try the following simple routine to get a better idea of scientific notation. Note that ZYBASIC requires numbers less than 1 to be entered with a leading zero.

```
10 INPUT X,,,,
20 PRINT X
30 GOTO 10
```

To demonstrate the need for this notation the next example is a small demonstration program to calculate "factorials". The factorial of 3, written 3!, is 3*2*1 ie. 6. Factorial 4 (4!) is 4*3*2*1, and so on for other numbers. Factorials are simple to calculate for very small numbers but rapidly get out of hand. One use for factorials is that the number of arrangements or "permutations" of a set of items is equal to the factorial of the number of items. For example, the number of possible seating arrangements of 20 people on 20 chairs is the huge number in the paragraph at the foot of the previous page. The following routine will calculate the factorial of other numbers of people X. (See how the computer will help you plan your next dinner party - such a boon to the busy hostess.)

```
100 INPUT X
110 A=1
120 FOR X=X TO 2 STEP -1
130 A=A*X
140 NEXT X
150 PRINT A
```

2.16 Numeric Variable Names

The following 260 names can be used for numeric variables, i.e symbolic storage locations for things which have specific numeric values.

```
A  thru Z
A1 thru Z1
A2 thru Z2
A3 thru Z3
A4 thru Z4
A5 thru Z5
A6 thru Z6
A7 thru Z7
A8 thru Z8
A9 thru Z9
```

These names can be used to contain any of the following data types :-

Decimal Integers: From -32767 to +32767.

Hexadecimal Integers: Up to four digits, prefixed by "#".

Floating Point Decimals: Between +-1.5E-39 and +-1.7E38.

(The above Data Types may be mixed in any one statement or line.)

Decimal fractions must be entered with a leading zero before the decimal point, i.e. 0.99 is legal, .99 is not.

Examples:

```
10 A = 56.3          ;A is set to numeric real 56.3

20 B = #F000         ;B is set to hexadecimal F000

30 C = 3             ;C is set to integer 3

40 X3 = 0.987        ;Notice the leading zero preceding the decimal
                      point; X3 = .987 is illegal.

50 C9 = 3+A*#AD      ;Mixed lines are valid. Here C9 is set to the
                      value of A times hexadecimal AD plus integer
                      value 3.

60 S4 = #F000+B      ;Here S4 is the sum of hexadecimal F000 and the
                      variable name B. Since the Interak VDU begins at
                      #F000, S4 in this example could represent a
                      visible location B spaces in from the beginning
                      of the VDU screen for direct transfer via a POKE
                      command.

70 POKE S4,Y         ;Here the data in Y is sent to the VDU address in
                      S4. See the POKE command for more on poking.
```

2.17 Numeric Operators

()	Parentheses
*	Multiply
/	Divide
-	Subtract
+	Add
>	Compare, Greater Than
<	Compare, Less Than
=	Compare, Equal To
<>	Compare, Not Equal To
<=	Compare, Less Than Or Equal To
>=	Compare, Greater Than Or Equal To

The result of any comparison is "1" if true, "0" if false. (See the description of the LET Command for more details).

Expressions are evaluated from left to right except that * and / are done first, then + and -, followed by any compare operators. Brackets may be used to alter the sequence of evaluation: the brackets are always evaluated before the other operators.

(see next page for examples)

Examples:

```
10 A = 3 * 4              ;A becomes 3 multiplied by 4

20 X4 = #F0*(S+V1)        ;X4 becomes the result of multiplying the
                           sum of S plus V1 by hex. F0.

30  X  =  ((A+B)*(C+D))/E  ;Here A is first added to B, and C to
                           D, then these two sums are multiplied
                           together and divided by E. The result is
                           put into X.
```

Logical OR and logical AND may be performed using :-

(var operator var) + (var operator var) is the OR function

(var operator var) * (var operator var) is the AND function

Examples:

```
10 IF (A<B)+(A>C) D=4     ;Here if A is less than B OR if A is
                           geater than C then D is set to 4.

20 IF (A<10)*(A>0) X=1    ;Here if A is less than 10 AND A is
                           greater than 0 then X is set to 1.
```

2.18 String Variables

26 string variables are allowed, and are named A$ thru Z$. Unlike the numeric variables which are storage locations for numbers, the string variables are storage locations for "strings" of characters which are to be used or processed as they stand, i.e. which do not represent numeric values.

The string variables are held in fixed memory locations, and are therefore accessable with other commands in the Basic set such as PEEK and POKE.

(See next page for memory allocations for string variables)

Memory allocations for string variables

	Hexadecimal	Decimal
A$	8100 ~ 81FF	33024 ~ 33279
B$	8200 ~ 82FF	33280 ~ 33535
C$	8300 ~ 83FF	33536 ~ 33791
D$	8400 ~ 84FF	33792 ~ 34047
E$	8500 ~ 85FF	34048 ~ 34303
F$	8600 ~ 86FF	34304 ~ 34559
G$	8700 ~ 87FF	34560 ~ 34815
H$	8800 ~ 88FF	34816 ~ 35071
I$	8900 ~ 89FF	35072 ~ 35327
J$	8A00 ~ 8AFF	35328 ~ 35583
K$	8B00 ~ 8BFF	35584 ~ 35839
L$	8C00 ~ 8CFF	35840 ~ 36095
M$	8D00 ~ 8DFF	36096 ~ 36351
N$	8E00 ~ 8EFF	36352 ~ 36607
O$	8F00 ~ 8FFF	36608 ~ 36863
P$	9000 ~ 90FF	36864 ~ 37119
Q$	9100 ~ 91FF	37120 ~ 37375
R$	9200 ~ 92FF	37376 ~ 37631
S$	9300 ~ 93FF	37632 ~ 37887
T$	9400 ~ 94FF	37888 ~ 38143
U$	9500 ~ 95FF	38144 ~ 38399
V$	9600 ~ 96FF	38400 ~ 38655
W$	9700 ~ 97FF	38656 ~ 38911
X$	9800 ~ 98FF	38912 ~ 39167
Y$	9900 ~ 99FF	39168 ~ 39423
Z$	9A00 ~ 9AFF	39424 ~ 39679

Locations 8000 ~ 80FF hexadecimal are reserved by ZYBASIC as a buffer area for internal string handling.

Although each string space is 256 bytes long a string may only be 255 bytes long. This is because each string is terminated by hexadecimal FF in the last used position plus one. i.e the usable space is 256 minus 1, =255.

Examples of string allocation:

```
10 A$ = "FRED"      ;The A string is set to FRED. Notice the quote
                     marks to enclose the string.

20 B$ = A$+"BILL"   ;B string becomes FREDBILL. See next page
                     "String Operators" for string arithmetic.
```

(See next page for string operators)

2.19 String Operators

ZYBASIC strings are defined and operated on by the following four symbol types:

""	String delimiters
+	Concatenation of strings.
=	Compare equal to
<>	Compare not equal to

"" The delimiters or quote marks are used to signify that the enclosed text is a string. Any character that can be enclosed in this way can be part of a string, and can be assigned a string variable name.

```
e.g.   10 A$ = "This is a string."
```

+ The plus symbol when used between two strings signify those two strings are to be concatenated. The effect of such an instruction is to form a new string which is the concatenation of the previous two. The original two strings will not be altered.

```
e.g.  10 A$ = "AAA"
      20 B$ = "BBB"
      30 C$ = A$+B$  ;Here C$ becomes AAABBB
      40 PRINT A$
      50 PRINT B$
      60 PRINT C$

e.g.  10 A$ = "ZY"
      20 B$ = "BASIC"
      30 PRINT A$+B$ ;Printing ZYBASIC
```

= The equals symbol when used between two strings causes the two strings to be compared for equality. This is carried out character by character, such that if both strings are identical the compare is considered true.

```
e.g.  10 A$ = "ABC"
      20 B$ = "ABC"
      30 IF A$=B$ PRINT "A$=B$"
```

<> The not equal symbol compares two strings for non equal characters and if found then the compare is true.

```
e.g.   10 A$ = "ABX"
       20 B$ = "ABC"
       30 IF A$<>B$ PRINT "A$<>B"
```

2.20 Keywords

Abbreviations used in Keyword descriptions:

```
exp......any algebraic equation, variable or number.
int......as for expr but only uses the integer component.
num......number
var......any variable name.
str......any string or string variable
```

List of Keywords:

```
@(int)..............returns contents of array location int.

ABS (exp)............returns absolute value.
AUTO int,int.........do auto line numbering.

BYE..................return to ZYMON.

CALL int.............do machine code routine.
CLS..................clear screen.
COS (exp)............returns cosine, exp in degrees.
COSR (exp)...........returns cosine, exp in radians.

DATA int,int,int,.....data for program use.
DON..................cursor line dots on.
DOFF.................cursor line dots off.

ED num...............edit line.

FOR exp TO exp........loop
FREE.................show spare RAM size.

GOTO int.............unconditional branch.
GOSUB int............unconditional call

IF exp exp...........conditional do, THEN is assumed.
IN var...............get port data.
INKEY var............get next key, else zero.
INPUT var............collect input data.
str = INPUT$.........get a string from the keyboard.
INT (exp)............returns the integer component.

LEN(str).............returns the length of a string.
LET str..............assign string var.
LET var=exp..........set var to exp, can be defaulted.
LINE int.............set VDU top or scroll point.
LIST int,int.........display program lines.

MID$(str,exp,exp).....substring function.

NEW..................open a new file.
NEXT var.............loop again.
```

```
OLD.................recover the old file.
ON var GOTO int.....size conditional branch.
ON var GOSUB int,int..size conditional call.
OUT port,data,data,...write to ports, incremental.

PAGE................set page mode.
PEEK (int)..........reads store contents.
POINT (int,int).....returns pixel condition or ASCII.
POKE addr,data,data,..write to store, incremental.
PRINT or ?..........output data to screen.
PROFF...............turn printer output off.
PRON................turn printer output on.

READ var............get data.
REM or ! ...........comment.
RESET (int,int).....reset pixel x,y. VDU K only.
RESTORE.............point to beginning of DATA.
RETURN..............from subroutine.
RND (int)...........random number. If int=0 returns 6 digit
                    fraction.
RUN.................execute a program.

SAVE................a program to tape.
SCROLL..............set scroll mode, defaults at NEW.
SET (int,int).......set pixel x,y. VDU-K only.
STEP int............loop advance.
SINR (exp)..........returns sine, exp in radians.
SIN (exp)...........returns sine, exp in degrees.
STOP................end process.
SQR (exp)...........returns the root.

TAB(int)............move print head to column specified. (1-255)

USR int.............do machine code routine.
```

The following useful functions are not implemented as they stand, but are easily derived:

```
LEFT$(str,exp).....derived from MID$(str,1,exp)
RIGHT$(str,exp)....derived from MID$(str,LEN(str)-exp,exp)
```

LOAD tape data. An implied command. There is no explicit tape load command; data is input from tape simply by turning the tape machine on and playing the tape. At the begining of such a load a certain amount of rubbish will be seen on the video display. This will be ignored by the computer and should be disregarded. ZYBASIC does not take advantage of the motor control and dual tape cassette arrangements which are present on the DTI-1 tape interface, but there is nothing to stop the user employing them in his own programs.

DETAILED COMMAND DESCRIPTION

In the following section each KEYWORD that can be understood by ZYBASIC is described in detail. Note that KEYWORDS have to be entered in UPPER CASE (i.e. in capital letters).

The keywords are arranged in alphabetical order and each description begins on a new page. The keyword is printed at the top of each page, at both left and right for quick reference.

(The more "grown up" way of dividing the keywords into groups, according to whether they are functions or commands and statements, has not been followed here. This is because it is mainly newcomers who need most to refer to the manual, and these are the very people who often don't really know in which group a keyword is to be found.)

In most cases examples of the use of the keywords are given; these examples are meant to serve only the purpose of example - where the example is a small program it is therefore not necessarily the quickest or best way to execute a task, it is only demonstrating a particular point.

The Interaktion User Group Newsletter will be the place where programming hints and tips can be published; some extracts from the newsletter are given in Appendix C of this manual.

@ @

Syntax:- var = @(num)
var = @(var)
var = @(exp)
@(num) = numb
@(num) = var
@(num) = exp
@(exp) = numb
@(exp) = var
@(exp) = exp

Allows reading or writing to the (single dimensioned) array.

10 A=@(3) Sets variable A to the value of array location 3.

10 A=@(C) The variable A is set to the value of the array location held in variable C.

10 B=@(X+Y) The variable B is set to the value of the array location held in the result of the expression X+Y.

10 @(3)=5 Array location 3 has the number 5 placed in it.

10 @(3)=A Array location 3 receives the number held in A.

10 @(3)=X*Y Array location 3 receives the number produced by X*Y.

10 @(X+Y)=3 The array location (X plus Y) receives the number 3.

10 @(Z+W)=D The array location Z plus W receives the number held in variable D.

10 @(X+Y)=X+Y The number X+Y is put into the array location X+Y.

Note:- ZYBASIC does not have a command to save an array on tape. If this is desired then a short tape save/load routine can be written using the ZYBASIC statements IN and OUT.

It is not entirely straightforward, since for example only one byte at a time can be saved to tape, but each element of the array is held in a four byte block. It is hoped that a fuller discussion can be made (after some thought) in the pages of the Interaktion Newsletter.

ABS ABS

Syntax:- var = ABS(numb)
var = ABS(var)
var = ABS(exp)

Returns the absolute or positive value of an expression.

(The keyword "ABS" may be replaced by "A.")

It is a function and may be embedded within an expression.

10 A = ABS(-3) A is set to 3.

10 A = ABS(B) A is set to the absolute value of B.

10 A = ABS(A) This form is useful to ensure that A is a positive or absolute number.

10 X = 72/ABS(D*E) This shows an example of the embedded syntax. Note that the brackets are required for the part of the expression relating to the ABS function.

10 X = 72/ABS(D)+E In this case the absolute value is taken of D on its own, then this value is added to E, and finally 72 is divided by the sum.

AUTO AUTO

Syntax:- AUTO num,num
AUTO num
AUTO ,num
AUTO

Automatic line numbering command.

(The keyword "AUTO" may be replaced by "A.")

A direct mode command, i.e. cannot be used in program statements.

Turns on the automatic line numbering function for convenient entry of programs - all you do is enter the actual program statements. You may specify the beginning line number and an increment to be used between line numbers. Both the line start and the increment can be defaulted in which case they auto-select to a value of 10. The AUTO functioon will preprint a new line number after each and every carriage return.

To exit from AUTO mode press CONTROL and C together.

AUTO 110,20 Sets the first program line to 110 and increases it by 20 after each carriage return.

AUTO 1000 Sets the first program line to 1000 and increases it by the default value of 10 after each carriage return.

AUTO ,20 Sets the first program line to the default value of 10 and increases it by 20 after each carriage return.

AUTO Sets the first program line to the default value of 10 and increases it by the default value of 10 after each carriage return.

Another use for AUTO which may not occur to you naturally, but which is perfectly orthodox, is for "zapping" unwanted groups of lines. For example if you have a subroutine at say 1000 which is seven lines long, and assume it has been written with line numbers which increase in the conventional steps of 10: all you have to do to remove these seven lines from your program is to type the following:-

AUTO 1000 and eight carriage returns.

(If you have a repeat key on your keyboard you can "zap" unwanted lines at a very quick rate, although unlike many home computer games no extra bonus points are awarded no matter how quickly you do your zapping.)

BYE BYE

Syntax:- BYE

Returns control to the ZYMON monitor.

(The keyword "BYE" may be replaced by "B.")

A direct mode command, i.e cannot be used in program statements.

BYE does not destroy the ZYBASIC program held in the text area. (See the OLD command for details of how to recover the program stored before the BYE command was used.)

Suppose for example that it was required to use ZYBASIC to generate a block of data which is to be stored on tape using the ZYMON monitor: the procedure would be something along these lines:-

NEW to initialise ZYBASIC.

(Now load the ZYBASIC program from tape.)

RUN to generate the data to be saved.
(In this example assume that the program ends with the resultant data stored, using POKEs, into addresses #0800 through #09FF.)

BYE to enter ZYMON.

S 0800 09FF ZYMON will save the data

E A000 to re-enter the ZYBASIC.

OLD The program is reinstated and ready to RUN again with more data.

An interesting refinement of this technique would be to BYE the ZYBASIC and tape save the entire store, S 1000 BFFF. You could then reload under ZYMON and having re-entered the ZYBASIC, E A000, you could type OLD and continue from where you left off. Remember to use GOTO line number rather than RUN to restart so that no parameters are altered in the previously saved program. See the RUN command for more details on starting with GOTO.

CALL CALL

Syntax:- CALL num
CALL var
CALL exp

Call to user subroutine.

(The keyword "CALL" may be replaced by "CA.")

CALL and USR are synonymous and enable a machine code subroutine to be called from within a ZYBASIC program. ZYBASIC responds to both keywords to improve program portability: CALL = USR = CALL; use the one you prefer.

The computer will execute code from the indicated address until a return (hex. C9) is encountered; execution will then return to ZYBASIC at the point it left off.

So that the return will be successful, and so that ZYBASIC will be in the same state as it was before the USR, it is vital that ZYBASIC's stack area is not altered by the user routine. Some user routines will not need to use a stack, but if yours does the recommended way to avoid disturbing ZYBASIC's stack is to leave it strictly alone and to use a different one for your subroutine.)

Once you change to a different stack you can no longer use the simple RET (hex. C9) machine-code instruction to get back. In these circumstances another method can be used to return control to ZYBASIC at the end of the CALLed machine code subroutine. The method which follows is based on the fact that immediately after the ZYBASIC CALL command is executed, (i.e on entry to a machine code subroutine), the BC register pair contains the required return address:

```
10 CALL #800      Start executing machine code from #0800
            !
            !
      0800  LD SP,0FFFH              Log in to a new stack
            PUSH BC                  Save return address
            ......
            Move the space invaders      (Example user
            Bleep the sound card            routine)
            Update deadmen counters
            ......
            !
            !
            POP HL                   Get return address
            JP (HL)                  Return to ZYBASIC
```

The following store areas are not used by ZYBASIC, and can be used for your own machine code programs, data, etc.:-

0800 ~ 0FFF hex. ...not used by ZYBASIC
9B00 ~ 9FFF hex. ...not used by ZYBASIC

(Description continues on next page)

CALL (continued from previous page) CALL

You may also use 1800 ~ 7FFF hex. but since this is the area where the program text is stored the amount of space available here depends on the size of your ZYBASIC program. Unless the ZYBASIC program is very big there will generally be a few K spare at the top of the text area. (You can find out by using the FREE command after loading the ZYBASIC program.) Note that the CALLed subroutine must not unintentionally alter ZYBASIC or the program text area. If it does, hard to predict results would be obtained on the return to the corrupted ZYBASIC and its program.

```
      0000 - 07FF  ZYMON
free..0800 - 0FFF
      1000 - 1800  ZYBASIC numeric variables, stack and spad
      1800 - 7FFF  ZYBASIC program text area, user code
      8000 - 9AFF  ZYBASIC string variables
free..9B00 - 9FFF
      A000 - BFFF  ZYBASIC interpreter
```

CLS CLS

Syntax:- CLS

Causes the VDU screen to cleared.

(The keyword "CLS" may be replaced by "C.")

10 CLS The screen is cleared unconditionally.

10 IF V=567 CLS If the variable V is equal to 567 then the VDU screen is cleared.

NOTES:-

If you use a VDU-K

00 hex. is written to all screen positions. This is to cause a graphics clear function such that POINT commands will return the pixel unset condition.

If you use a VDU A,B,G 3-card set (predecessor to the VDU-K):

20 hex., the ASCII space code, is written to all screen positions. In this case pixel representation is beyond the ability of the hardware, and cannot be utilised.

If the A,B,G cards have been altered to carry unspecified RAM chips then the auto selection process may not function corectly. The test mechanism used to decide which VDU is in the system depends on the design of the two VDUs. The test is based on the fact that the early A,B,G set had 768 bytes of RAM whilst the present VDU-K has 1024 bytes of RAM. If for some reason this has been altered the auto screen select may fail.

The CLS function will also work correctly on VDU-2K modified cards.

COS COS

Syntax:- COS(num)
COS(var)
COS(exp)

Returns the cosine of an angle which is expressed in degrees.

(There is no abbreviation for the keyword "COS".)

It is a function and thus may be embedded within an expression.

10 A = COS(30) A is the cosine of 30 degrees

10 A = COS(30)+COS(60) A is set to the cosine of 30 degrees plus the cosine of 60 degrees.

10 PRINT COS(A) This will print the cosine of the variable A, where A is expressed in degrees.

10 Y = COS(X+3.3) Producing the cosine of the sum of the two angles, X and 3.3, expressed in degrees.

10 IF COS(A+X) = COS(A-X) GOTO 50 This uses the COS function in a conditional branch to decide the outcome of some calculation.

Try these examples:-

```
1  REM PRINT A TABLE OF COSINES
10 CLS:DOFF:PAGE:LINE 1          Initialise
50 PRINT "Angle....Cosine"       Print a title
60 FOR X=10 TO 180 STEP 10       For X begins at 10
70 PRINT %4,X,,,,,%1,COS(X)      Print degrees and cosine
80 NEXT X                        Loop if X not 190
90 STOP                          End

1  REM PRINT A COSINE WAVE
10 CLS:DOFF:PAGE:LINE1           Initialise
20 FOR X=1 TO 63                 For X begins at 1
30 Y=24+(20*COS(X*10))           Produce cos wave point
40 SET (X,Y)                     Display graphic point
50 NEXT X                        Loop if X not 64
60 STOP                          End
```

Line 30 could have been SET(X,24+(20*COS(X*10))) which would eliminate the need for line 40 and speed up the display.

Angles greater than 360 degrees will be computed with reducing precision as the total number of degrees becomes large with respect to the first circle.

COSR COSR

Syntax:- COSR(num)
COSR(var)
COSR(exp)

Returns the cosine of an angle that was expressed in radians.

(The keyword "COSR" may be replaced by "C.")

It is a function and thus may be embedded within an expression.

```
10 A = COSR(3)
```
A is the cosine of 3 radians.

```
10 A = COSR(3)+COSR(6)
```
A is set to the cosine of 3 radians plus the cosine of 6 radians.

```
10 PRINT COSR(A)
```
This will print the cosine of the variable A, where A is expressed in radians.

```
10 Y = COSR(X+3.3)
```
Produces the cosine of the sum of the two angles, X and 3.3, expressed in radians.

```
10 IF COSR(A+X) = COSR(A-X) GOTO 50
```
This uses the COSR function in a conditional branch to decide the outcome of some calculation.

Notes:-
A complete circle of 360 degrees is expressed as two pi radians where pi is 3.14159. So if P = 3.14159 = pi then:

```
 90 degrees = 0.5 * P = 1.57079 rads or pi/2 rads
180 degrees = 1.0 * P = 3.14159 rads or pi   rads
270 degrees = 1.5 * P = 4.71239 rads or pi*(3/2) rads
360 degrees = 2.0 * P = 6.28319 rads or pi*2 rads
```

As 360 degs = 6.28319 radians then 1 radian = 360/6.28319, giving 1 radian = 57.29574 degrees.

Also 1 degree = 6.28319 rads/360 = 0.01745, giving 1 degree = 0.01745 radians.

Angles greater than 6.28319 radians will be computed with reducing precision as the total number of radians becomes large with respect to the first circle.

DATA DATA

Syntax:- DATA num,num,num,num,num,num,num
DATA var,var,var,var,var,var,var
DATA str,str,str,str,str,str,str
DATA num,var,str,num,var,str,str
DATA var,num,num,str,var,num,str

Allows DATA to be stored within a program in a form which can be accessed by a READ statement.

(The keyword "DATA" may be replaced by "D.")

The DATA items will be read sequentially, starting with the first item in the first DATA statement, and ending with the last item in the last DATA statement. Items in a DATA statement may be strings, numbers, string variables or numeric variables.

Expressions cannot be included within a DATA statement.

Each item is separated from the next by a comma.

The DATA statement is terminated by a carriage return.

```
10 DATA 1,2,3,4,5,6,7,8
```

```
10 DATA A,B,C,D,E,F
```

```
10 DATA A$,B$,C$,D$,"FRED","BILL"
```

```
10 DATA #F000,#F001,#F002,#ABCD
```

```
10 DATA 234,A,D$,#F000,"FRED",3.142
```

A DATA item can only be READ into the same variable name type. i.e. String data can only be READ into string variables and numeric data READ into numeric variables.

Try this program:-

```
5  RESTORE                  Reset the DATA pointer
10 CLS:DOFF:PAGE:LINE1      Initialise
20 FOR C=1 TO 18             For C begins at 1
30 READ X                     Get X coordinate
40 READ Y                     Get Y coordinate
50 SET (X,Y)                  Light pixel at X,Y
60 NEXT C                    Repeat till C is 19
70 STOP                     Program ends
80 DATA 1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10
90 DATA 11,11,12,12,13,13,14,14,15,15,16,16,17,17,18,18
```

DON DON

Syntax:- DON

Turns on the cursor marker dots.

(There is no abbreviation for the keyword "DON".)

When switched on a line of dots occupy the current line from the cursor position until the end of the line. These dots act as an aid in the correct counting of columns when using and planning the video display part of a program. Also when PAGE mode (q.v.) is selected the dots help the user find the position of the cursor on the screen. The dots may be switched on under program control or in the direct mode.

After completion of the NEW command, the DON function is called by the ZYBASIC interpreter and the dots are turned on.

10 DON The cursor line dots are turned on.

10 IF (A=2)+(B=3) DON If the variable A is equal to 2, or the variable B is equal to 3, then the cursor line dots are switched on.

DOFF DOFF

Syntax:- DOFF

This command turns off the cursor line dots.

(There is no abbreviation for the keyword "DOFF".)

```
10 DOFF              Turns the cursor dots OFF.

10 IF A$=B$ DOFF     If string variable A is equal to string variable B
then turn OFF the cursor line Dots
```

ED ED

Syntax:- ED line number Edit a program line.

This command is a direct mode command. It cannot be embedded within a program statement line. (There is no abbreviation for the keyword "ED".)

To edit a line, type EDline number CR. The line to be edited will be displayed. You can now use CTRL-N to get the next character and CTRL- R to get the rest of the line. By typing CTRL-N each character of the line will be displayed beneath the displayed entire line. You may now use CTRL- N to advance to a desired point and insert code by typing it in, or you may delete and/or correct errors by stepping past an error and backspacing it out, typing any desired new text before advancing further with CTRL-N. When all changes have been completed, CTRL-R will display the remainder of the line and a carriage return will log the new line into the text area. The EDited line that is finally displayed will not be logged into the text area unless the carriage return is pressed. If the line number of the EDited line is the same as any which already exists the EDited line will replace the original. CTRL-C will end EDit with no effect on the original line.

Examples.
Suppose 50 C = 3333 is to be changed to 50 C = 1333

Enter ED50 carriage return and the computer will display

```
50 C= 3333     This is the original line
50 _           Now type CTRL-N to get the C
50 C_          another CTRL-N to get the space
50 C _         another CTRL-N to get the =
50 C =_        another CTRL-N to get the 3, to be altered.
50 C = 3_      now DELETE the 3, or use CTRL-H, backspace
50 C = _       and type 1, which is the desired change.
50 C = 1_      finally type CTRL-R to get the rest of the line
50 C = 1333_   Terminate the session with carriage return.
```

Suppose you wish to alter a line number from 50 to 80. As before type ED50 carriage return.

```
50 C = 1333    The original line.
50 _           now DELETE or BACKSPACE
50_            again DELETE or BACKSPACE
5_             again DELETE or BACKSPACE
_              enter the 8
8_             enter the 0
80_            enter the space. (This is optional)
80 _           and get the rest with CTRL-R.
80 C = 4634_   Press carriage return to log in line 80
```

At this point both lines exist in store. That is:-

```
50 C = 1333
80 C = 1333    You may now delete the unwanted one by typing
50 carriage return.
```

FOR TO STEP....NEXT FOR TO STEP....NEXT

Syntax:- FOR variable = value TO value STEP value

NEXT variable

(The keywords "FOR", "STEP", "NEXT" may be replaced by "F.", "S.", "N.")

FOR opens a repetitive loop so that a sequence of program statements may be repeated a specified number of times. FOR loops may be nested one within the other. On the right of the equals sign any combination of var,exp or num can be used in place of value. The step value can be any combination of var,exp or num, but must be, or resolve to be, an integer value. The STEP value is optional and defaults to 1 if not entered.

A FOR loop works as follows:-
The first time a FOR statement is encountered the variable after the FOR keyword is set to the initial value, the first after the equals sign. Execution of program statements proceeds until a NEXT statement is encountered. At this point, the counter variable is incremented by the amount specified in the STEP value, default is one. Then the counter variable is compared to the value following the TO keyword. If the counter is greater than the TO value the loop is completed and execution continues through the NEXT keyword and on to the following statement. If the compared value is less than or equal to the TO value program execution continues from the line following the FOR line.

```
10 A = 10                        Set A to 10
20 B = 1                         Set B to 1
30 FOR X = B TO A STEP 2         Loop X is 1
40 PRINT X,                      Print value X no return
50 PRINT X*X                     Print value X*X
60 NEXT X                        Do until X greater than 10
70 STOP                          End
```

```
10 FOR X = 10 TO 1 STEP -1       Loop X is 10
20 PRINT X                       Print current X
30 NEXT X                        Print all X's 10-1
40 STOP                          End
```

```
5  CLS                           Clear the screen
10 FOR X = 0 TO 63               Get X 0-63
20 FOR Y = 0 TO RND(47)          Get Y 0-RND(47)
30 SET (X,Y)                     Set pixel X,Y
40 NEXT Y                        Set all Y's for this X
50 NEXT X                        Repeat for all X's
40 STOP                          End
```

Note that in keeping with a long standing convention, the FOR loop is always executed at least once. In many BASICs (including this one) a loop which begins say "10 FOR J = 100 TO 99" is executed once even though it starts with J = 100, i.e. already past the limit 99.

FREE FREE

Syntax:- keyword FREE
var = FREE

This is a function which acts as an argument to some other command keyword.

(The keyword "FREE" can be replaced by "F.")

If used within a statement line, FREE will produce as an integer the remaining free storage space available.

FREE can be included in an expression.

Examples.

```
10 PRINT FREE   Print remaining store size.
```

```
10 A=FREE
20 IF A < 6 PRINT "Out of space"
```

```
10 X=FREE
20 B=5234
30 IF X/4 < B PRINT "Your array is too big!"
```

```
10 PRINT FREE/4 + 324 + SIN(FREE)
```

```
10 X = RND(FREE)        This is a useful way to obtain a random number
                        which is fairly unpredictable.
```

GOTO GOTO

Syntax:- GOTO line number
GOTO var
GOTO exp
GOTO int

This keyword transfers control to the specified line number.

(The keyword "GOTO" may be replaced by "G.")

Used alone GOTO line number results in an unconditional branch. Test statement lines may precede the GOTO to effect a conditional branch.

```
10 GOTO 10
```
An infinite loop. (Use CTRL-S followed by CTRL-C to break out of the loop - the use of the System Reset switch will also do it, but is a bit drastic.)

```
10 A = #F000
20 B = RND(256)
30 POKE A,B
40 GOTO 20
```
VDU pointer
Random data
Data to the screen
Repeat forever with different data. (Get out as before - CTRL-S then CTRL-C)

```
10 CLS
20 FOR A = #F000 TO #F000 + 24*32 STEP RND(128)
30 POKE A,X
40 NEXT A
50 X = RND(128)
60 GOTO 20
```
Clear screen
From VDU start to finish, in random size steps
Poke this screen location with X
Increase A by the random step
Random data
Repeat forever. (Until you get out)

```
10 A = #F000
20 C = 24*32
30 B = RND(128)
40 POKE A,B
50 C=C-1
55 A=A+1
60 IF C<>0 GOTO 30
70 GOTO 10
```
A points to #F000, the VDU screen RAM
Counter for 24 lines of 32 characters
Random data
Data to the screen
Decrement the counter C
Increment the screen address A
If not all done loop for another.
Do it all over again

(The last example would normally use a FOR-NEXT loop; the conditional GOTO at line 60 has only been incorporated as an easy to understand example of its use.)

You can use GOTO in command mode as an alternative to RUN: GOTO line number causes execution to begin at the specified line number, without the automatic clear of the variable names space. This allows you to pass values assigned in the command mode to variables in the execute mode.

GOSUB GOSUB

Syntax:- GOSUB line number
GOSUB var (this should be an integer value)
GOSUB exp (this must resolve to an integer)
GOSUB int

(The keyword "GOSUB" may be replaced by "GOS.")

This command makes an internal note of the next instruction to be executed, then transfers control to the ZYBASIC language subroutine which is specified by the stated line number. When a RETURN statement is encountered, control returns to the statement following the GOSUB which called the subroutine. A conditional call can be effected by including a test ("IF") statement before the GOSUB. Any subroutine can itself contain a GOSUB to another subroutine, i.e. GOSUBs may be nested within themselves.

```
2 PAGE: LINE1: CLS: DOFF
5 PRINT " A A*A A*A*A A*A*A*A"
10 FOR A = 1 TO 10
20 GOSUB 100
30 NEXT A
40 STOP
99     REM  Subroutine 100 follows
100 PRINT A,,,A*A
110 RETURN
```

```
10 INPUT "TYPE A NUMBER 1 TO 3",A       Get input
20 IF (A<1)+(A>3) GOTO 10             If out of range try again
30 A=INT(A*100)                     Make 1 ~ 3 into 100 ~ 300
40 GOSUB A                        GOSUB 100 ~ 300
50 GOTO 10                      Repeat forever
99     REM  Subroutine 100 follows
100 PRINT "1 2 3": RETURN
199    REM  Subroutine 200 follows
200 PRINT "2 3 1": RETURN
299    REM  Subroutine 300 follows
300 PRINT "3 2 1": RETURN
```

In the second example you could discard line 30 and make line 40 GOSUB INT(A*100); this would leave unchanged the number which the user input for A. It could then be used for other tests by routines elsewhere in the program.

It is good programming "manners" to introduce a subroutine with a REM statement as in the above examples, and it makes for a more readable program if you have just one RETURN per subroutine, positioned as its last statement. (If memory space is limited, or if speed of execution is critical, these formalities may be disregarded.)

Note: Return from a subroutine must be in an orderly manner so that ZYBASIC can keep control of the execution of the program, restore the stack, and so on. It is very disorderly to leave a subroutine by means of a GOTO on its own: always GOTO a RETURN statement to leave.

IF IF

Syntax:- IF case execute

(There is no abbreviation for the keyword "IF".)

"Case" can be any compare expression.
e.g.
```
IF A = B
IF A+C/3 = D*F/S
IF A$ = "FRED"
```

"Execute" can be any valid ZYBASIC command keyword or an expression.
e.g.
```
IF A = B PRINT "XYZ"
IF A+C/3 = D*F/S GOTO 60
IF A$ = "FRED" A=A*A
```

The IF keyword instructs the computer to test if the logical or relational expression following the IF keyword is true or false. If the test is true the remainder of the line is executed. If the test is false the remainder of the line is skipped and the next line in sequence is executed. As ZYBASIC permits multiple statements on a single line the IF construction can be used to decide whether or not several statements which may follow on the same program line are executed. If the result of the IF test is false all of the remaining statements in that line are abandoned; if it is true they are may all be executed. The second of the following examples demonstrates this.

```
10 PRINT "Enter your name "
20 A$=INPUT$
30 IF MID$(A$,1,1) = "B" GOTO 100
40 PRINT "HELLO ",A$
50 STOP
100 PRINT "MY NAME STARTS WITH B ALSO"
110 GOTO 40
```

```
10 IF A>127 PRINT "OUT OF RANGE":GOTO 200
20 GOTO 50
```

Note that if the condition is not met then none of the rest of the line is executed: in the last example if the condition in line 10 is not met executions jumps straight to line 20.

```
5  A=RND(7)-1:Y=RND(7)-1
10 IF A=0 PRINT "FUEL IS GONE": H=250: GOSUB 100: GOTO 30
20 IF A>1 PRINT "YOU HAVE LANDED O.K.": GOTO 500
30 PRINT "YOU CRASHED": IF Y=0 PRINT "YOU ARE ALIVE": GOTO 500
40 PRINT "YOU ARE DEAD": GOTO 500
100 PRINT "AHHHHH": RETURN
500 FOR X=1 TO 1500: NEXT X: REM (This is just a delay)
510 PRINT
520 GOTO 5
```

(examples continue on the next page)

IF (continued from previous page) IF

(Boolean equations)

The IF command allows the Boolean logic operators "OR" and "AND" to be simulated, as the logic they follow is implicit in the logic of the IF command:

For "OR" use "+"
For "AND" use "*"

e.g. To code the statement

"If X=0 or X=1, and if Y=0 or Y=1 then go to line 80"

within a ZYBASIC program, the "+" and "*" operators are used:

```
IF ((X=0) + (X=1)) * ((Y=0) + (Y=1)) GOTO 80
```

This is used as line 60 in the program below.

Example:

```
10 CLS: PAGE: LINE1: DOFF
20 PRINT "ENTER A BINARY DIGIT, 0 OR 1."
30 INPUT X
40 PRINT "ENTER ANOTHER BINARY DIGIT. "
50 INPUT Y
60 IF ((X=0) + (X=1)) * ((Y=0) + (Y=1)) GOTO 80
70 PRINT "INVALID INPUT": PRINT: GOTO 20
80 PRINT: PRINT %1,X,%1,Y,,"=",%1,X*2+Y
90 PRINT: GOTO 20
```

IN IN

Syntax:- var = IN(port)

Get port data to variable

(There is no abbreviation for the keyword "IN".)

This command allows a variable (A thru X9) to receive the numeric value of the data input at any of the Z80's 256 ports. Since the data ports are 8-bit, the value of the data will lie in the range 0 to 255 (decimal) i.e. 0 to #FF (hex.)

Example 1

```
10 CLS: PAGE: LINE1
20 PRINT"   KEEP PRESSING KEYS"
30 SCROLL: LINE 20
40 A=IN(#40): IF A=0 GOTO 30
50 POKE #F000+32+A, #1C
60 PRINT "KEY VALUE IS", %4,A
70 GOTO 40
```

Example 2

```
10  REM ************************
20  REM    MAIN PROGRAM BEGINS
30  REM ************************
40  FOR A=1 TO 32
50    B=#41
60    GOSUB 100
70  NEXT A
80  STOP
97  REM ************************
98  REM  PRINTER OUTPUT ROUTINE
99  REM ************************
100 S=IN(6):              REM Get printer status (Port 6)
110 IF S<#7F GOTO 100:    REM If busy loop till free
120 OUT(7),B:             REM Print B data (Port 7)
130 RETURN
```

INKEY INKEY

Syntax:- INKEY var

Returns the value of the last key to be pressed on the keyboard.

(The keyword "INKEY" may be replaced by "INK.")

This command takes the same form as the INPUT command (q.v.), however, the program is not halted while waiting for an input. If a key is being pressed on execution of this command, the specified variable will be set to the value of the key pressed. If no key is being pressed during an INKEY instruction the variable will be set to zero. ZYBASIC does not wait for a key to be pressed - it only returns whatever it found when it looked. (If the LKP-1 latched keyboard port card is used for the keyboard interface then a keystroke made any time before ZYBASIC looked will be held latched on the card ready for ZYBASIC to take it. The hardware design of the LKP-1 card makes sure that the latch is cleared after it is read, so the same keystroke will not be read again.)

```
10 CLS: DOFF: LINE 22: SCROLL
20 INKEY X
30 IF X=0 GOTO 20
40 PRINT "KEY NUMERIC VALUE WAS",%1,X
50 POKE #F130,X
60 GOTO 20
```

In the above program line 40 prints the value of the key pressed and line 50 reproduces the corresponding character at the centre of the screen.

```
1000 PRINT "Press any key to start."    Request data
1010 INKEY K                            Get data
1020 IF K=0 GOTO 1010                   Loop if no data
1029 REM K HAS A VALID KEY CODE         Drop through with the key

10  PRINT "Type YES or NO (Y/N)"        Request input
20  INKEY K                             Get input (ASCII "N" = #4E)
30  IF K =  #4E GOTO 800                Go  TO  800 if N typed
40  IF K <> #59 GOTO 20                 If not Y or N (ASCII "A" = #59)
50  PRINT "Y RECEIVED": GOTO 10         Y route
800 PRINT "N RECEIVED": GOTO 10         N route

10 PRINT "Enter E,D,O or P"             Data request
20 INKEY K                              Read data given
30 IF K=69 E=1:GOTO 70                  If E key seen jump on
40 IF K=68 E=2:GOTO 70                  If D key seen jump on
50 IF K=79 E=3:GOTO 70                  If O key seen jump on
60 IF K=80 E=4:GOTO 70                  If P key seen jump on
65 GOTO 500                             Exit key not recognised
70 ON E GOTO 100,200,300,400            Branch on key
```

(If you are confused by the numbers in lines 30 to 60 of the last example note that decimal 69 is #45, i.e. the ASCII code for "E", similarly 68, 79, 80, are #44, #4F, #50, i.e. "D", "O", "P", respectively.)

INPUT INPUT

Syntax:- INPUT var
INPUT "message" var
INPUT var,var,var,var
INPUT "message"var,var,var,var

Collects a numeric string typed on the keyboard. Waits until the string is complete (signified by a carriage return) before proceeding.

(The keyword "INPUT" may be replaced by "IN.")

This can be used to set a variable or array location to a value typed on The keyboard. e.g. Consider the line "10 INPUT B". When the program reaches this line it will print B and wait for a value to be typed on the keyboard (followed by carriage return). The number typed will be assigned to B. The value entered from the keyboard can be an expression. e.g. 12*X+1. If the expression is invalid then ZYBASIC will reprint the prompt i.e. B, and wait for a further input. Alternatively, the INPUT command can print a message before waiting for an input. e.g.

```
10 INPUT "HOW MANY FOR DINNER?"D,"FOR TEA?"T
```

This will result in the question, HOW MANY FOR DINNER?, if an invalid answer is given, HOW MANY FOR DINNER?, will be repeated. If the answer was valid the value will be assigned to D and FOR TEA? will be printed. The next valid input will be assigned to the variable T. Only part of a prompt need be repeated when an invalid answer is input. This is achieved by splitting the prompt with further quotation marks (quotes.)

e.g.

```
20 INPUT "HOW MANY","FOR DINNER"?"D
```

Only the FOR DINNER? will be repeated.

Note: A comma after the input "string" will cause the variable name to be printed, e.g. 10 INPUT "ENTER DATA",A will produce ENTER DATA A_ whereas no comma causes no name to be printed, e.g. 10 INPUT "ENTER DATA"A will produce ENTER DATA_

There is one exception to the rule that an invalid answer causes the prompt to be reissued: if the user types CTRL-C in response to the input prompt the program running will be abandoned and command returned to the Direct Mode. (Without this feature it would be impossible to break out of a program which had reached an INPUT statement, other than by the use of the System Reset or something equally drastic.)

(Example program on next page)

INPUT (continued) INPUT

Example of use of INPUT command:

```
10 PAGE: LINE 1: DOFF: CLS          Initialise
20 SCROLL: LINE 22                  Scroll 22, 23, 24 only
30 INPUT "ENTER X (0-63)"X          Get X coordinate
40 IF (X<0)+(X>63) GOTO 30          If out of range re-get X
50 INPUT "ENTER Y (6-47)"Y          Get Y coordinate
60 IF (Y<6)+(Y>47) GOTO 50          If out of range re-get Y
70 SET (X,Y)                        Set pixel
80 PRINT: PRINT                     Scroll up
90 GOTO 30                          Repeat this forever
```

INPUT$ INPUT$

Syntax:- var$ = INPUT$

This instruction will input a string from the keyboard. (Terminated by carriage return).

(There is no abbreviation for the keyword "INPUT$".)

Any character which can be typed on the keyboard can be part of an INPUT string, and can be assigned to a string variable name.

When a ZYBASIC statement which contains the instruction keyword INPUT$ is encountered execution will halt temporarily. A cursor will be shown at the next print position and any data typed until carriage return will form a string. CTRL-C will cause the program to terminate, and backspace and delete will work in the normal way

```
10 PRINT "Enter your name"              Request input data
20 A$ = INPUT$                          Collect input data
30 PRINT "Your name is",A$              Print message and input data
40 STOP                                 End
```

The next example shows how INPUT$ may be used to obtain data and then process it to obtain results based upon that data.

```
10 PRINT "ENTER YOUR NAME"              Request input data
20 A$=INPUT$                            A$ holds input data
30 A=LEN(A$)                            A holds input data length
40 PRINT A$,"HAS ",%1,A,"LETTERS IN IT" Say its length
50 STOP                                 End
```

(The next example has been printed on the following page as it is too long to print on this page without splitting it in the middle.)

(Command description continues on next page)

Example program

```
7         REM   * * * * * * * * * * * * * * *
8         REM   *   REQUEST USER INPUT     *
9         REM   * * * * * * * * * * * * * * *
10  PRINT "PLEASE ANSWER YES OR NO"
20  C$ = INPUT$
26        REM   * * * * * * * * * * * * * * *
27        REM   *  ACCEPT UPPER OR LOWER  *
28        REM   *       CASE RESPONSE      *
29        REM   * * * * * * * * * * * * * * *
30  IF (C$ = "yes") GOTO 500
40  IF (C$ = "YES") GOTO 500
50  IF (C$ = "no")  GOTO 300
60  IF (C$ = "NO")  GOTO 300
67        REM   * * * * * * * * * * * * * * *
68        REM   * INVALID INPUT DETECTED  *
69        REM   * * * * * * * * * * * * * * *
70  PRINT "RESPONSE DOES NOT COMPUTE - PLEASE RETRY."
80  GOTO 10
297       REM   * * * * * * * * * * * * * * *
298       REM   *  FOLLOW NEGATIVE PATH    *
299       REM   * * * * * * * * * * * * * * *
300 PRINT "NEGATIVE RESPONSE.": STOP
497       REM   * * * * * * * * * * * * * * *
498       REM   *  FOLLOW POSITIVE PATH    *
499       REM   * * * * * * * * * * * * * * *
500 PRINT "POSITIVE RESPONSE.": STOP
```

In the above example a choice is required of the user. If an illegal reply is given the program issues a suitable response and retries for input. If you have studied the section of this manual which describes the IF command and Boolean logic operators you have thought that lines 30 to 60 in the above program could be condensed as follows:

```
30  IF (C$ = "yes")+(C$ = "YES") GOTO 500
50  IF (C$ = "no") +(C$ = "NO")  GOTO 300
```

Unfortunately in this version of BASIC the Boolean logic operation cannot be applied to string comparisons, so the upper and lower case conditions above have to be written out individually.

INT INT

Syntax:- INT(exp)
INT(var)
INT(num)

This is a function and is returns the integer value of the expression, variable or number that is within the brackets.

(There is no abbreviation for the keyword "INT".)

```
10 A = INT(3.34)
```
A is set to the integer result 3.

```
10 A = INT(3.99)
```
A is set to the integer result 3.

```
10 A = INT(A)
```

```
10 A = INT(B*C/3+4)
```

```
10 A=COS(INT(SQR(C)))
```
A is set to the cosine of the integer value of the square root of C.

```
10 D = 3.999
20 FOR A = 1 TO 50 STEP INT(D)
30 PRINT A,D
40 NEXT A
```

INT(D) in line 20 of the last example ensures satisfaction of the requirement that the step in the FOR-NEXT loop be an integer; as it happens, ZYBASIC will do this anyway, even if the user forgets - to verify this replace line 20 above with 20 FOR A = 1 TO 50 STEP D

LEN LEN

Syntax:- LEN(string)

This is a function which returns the numeric length of a given string.

(There is no abbreviation for the keyword "LEN".)

The result of LEN(string) is placed in or used by numeric type variables. A string variable cannot recive the result. i.e. A = LEN(B$) is legal as A receives the numeric length of B$ but A$ = LEN(B$) is illegal and should not be used.

```
10 C$="BOB"
30 Y=LEN(C$)  The variable Y will be set to 3, the length of "BOB".
```

```
10 C$="Llanfairpwllgwyngyllgogerychwyrndrobwllllantysyliogogogoch"
30 W=LEN(C$)  The variable W will be set to 58, the length of "Llanfair-
pwllgwyngyllgogerychwyrndrobwllllantysyliogogogoch".
```

```
10 DOFF:CLS:LINE 1
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 FOR X=1 TO LEN(A$)
40 PRINT MID$(A$,1,X)
50 NEXT X
```

Related Note: RIGHT$ AND LEFT$

These two commands do not exist in ZYBASIC; however this is no hardship as they can easily be simulated by the correct use of MID$. e.g.

LEFT$(str,exp) derived from MID$(str,1,exp)

RIGHT$(str,exp) derived from MID$(str,LEN(str)-exp,exp)

LET LET

Syntax:- LET var = num
LET var = exp
LET var = var

The LET command is used to give a value to a variable. The keyword "LET" can be implied by the statement, i.e. the word "LET" may be omitted.

(There is no abbreviation for the keyword "LET".)

10 LET A=2 Here A is set to be 2.

20 B=4 Here B is set to be 4. Notice that "LET", which was used in the previous example, has been omitted. In ZYBASIC "LET" can be used or not; entirely at the user's discretion.

30 C=A*A+6/SQR(B) Complex expresions will first be resolved and then the variable to the left ("C" in this example) will take on the value of the result.

Note that the equals sign is not used in the mathematical sense (as a statement of the equality of two or more quantities). In mathematics, it is obviously nonsense to write X = X + 1, but it is a perfectly legal statement in the BASIC language. In BASIC the equals sign serves a different purpose: it stands for "is replaced by", so that for example X = X + 1 means the value of X before this statement was encountered is to be removed and replaced by the old value of X plus 1.

LET($) LET($)

Syntax:- LET str=str

Here the LET command is used to assign a string to a string variable. The keyword "LET" can be implied by the statement, i.e. the word "LET" may be omitted.

(There is no abbreviation for the keyword "LET".)

```
10 LET A$="FRED"        Here A$ is set to be the string "FRED"
```

```
10 A$="FRED"            This is the same as example 1 except that the word
```
"LET" has been omitted. It is up to the user whether he types the word "LET" - some people think it makes a program more readable, others don't. If you don't care, it is best to leave it out as is then only wasting memory space.

```
10 A$="FRED"            A$ is set to be "FRED"
20 B$=A$                B$ is set to be as A$.
```

```
10 A$="FRED"            A$ is set to "FRED"
20 B$=MID$(A$,1,2)      B$ is set to be "FR", a small part of A$
```

Related Note: RIGHT$ AND LEFT$

These two commands do not exist in ZYBASIC; however this is no hardship as they can easily be simulated by the correct use of MID$. e.g.

LEFT$(str,exp) derived from MID$(str,1,exp)

RIGHT$(str,exp) derived from MID$(str,LEN(str)-exp,exp)

LINE LINE

Syntax:- LINE num
LINE var
LINE exp

(There is no abbreviation for the keyword "LINE".)

This command deals with the use of the screen by the ZYBASIC output routines. The lines (rows) of characters are numbered from the top, starting with line 1 as the highest line on the screen.

LINE IN PAGE MODE (q.v.)

If the screen is in PAGE mode then the LINE command will specify the start of the next PRINT output command. e.g.

```
10 PAGE: CLS: DOFF
20 LINE 10
30 PRINT "THIS IS LINE 10"
40 LINE 5
50 PRINT "THIS IS LINE 5"
60 STOP
```

In the above, PAGE mode is set up. Then line 10 is specified as the line on which the next message is to be printed, finally the same for line 5.

LINE IN SCROLL MODE (q.v.)

If you are in SCROLL mode then the LINE command specifies the highest line to which the screen will scroll, e.g.

```
10 CLS: DOFF
20 SCROLL
30 LINE 20
40 FOR A=1 TO 100
50 PRINT "ONLY SCROLLING 24 TO 20"
60 NEXT A
70 STOP
```

Using this the top portion of the screen can be retained as a data area whilst the bottom area can be used for operator input/output. For example, one very good use for this would be a short range scan display in the game STARTREK, so that status updates would not destroy the main view screen display.

You can of course switch between modes during program execution and so allow very complex displays to be produced. Perhaps you have noticed the use of LINE in the MONSTER MASH program (towards the end of this manual). This uses simple PRINT statements mixed with a fixed graphic display to give real time play with simple to program display of the score.

LIST LIST

Syntax:- LIST line number,number of lines

(The keyword LIST can be replaced by ".")

This can only be used as a direct command; it will print all the lines in the user's program if LIST carriage return is typed. The command can be abbreviated to a full stop. To print from a particular line number type LIST line number and carriage return. To print a small section of a program, e.g. from line no. 50 for 3 lines, type LIST 50,3 carriage return or .50,3 carriage return.

Examples

LIST	List all program lines.
LIST 100	List all lines from line 100 to the end of the program.
LIST 100,20	List twenty lines starting at line 100.

EXAMPLES with abbreviation "." used for "LIST"

.	List all program lines
.100	List all lines from line 100
.100,20	List twenty lines starting at line 100

To use list to print your programs on a hardcopy device enter PRON carriage return followed by full stop carriage return.

PRON <CR>	Turn on hardcopy
.<CR>	List all program to screen and printer.

Note that the syntax of this command is deliberately different from that used in some other BASICs (where for example LIST 10,20 means list all the lines from 10 to 20). In ZYBASIC this example would mean start at line 10 and list 20 lines. No matter what line numbers have been used after line 10, ZYBASIC will consistently list 20; the older (non-ZYBASIC) syntax would cause the BASIC to list a variable number, depending on how many line numbers happened to be employed between line 10 and line 20. The very fact that the user is wanting a LIST implies that he hasn't got one and therefore he may not know what range of line numbers to quote to fill his screen or piece of paper.

MID$ MID$

Syntax:- MID$(str,start,number) = str
str = MID$(str,start,number)
MID$(str,start,number) = MID$(str,start,number)

This is the substring function and is used to obtain parts or slices of any given string.

(There is no abbreviation for the keyword "MID$".)

MID$ TO THE RIGHT OF THE EQUALS SIGN.

This allows a string variable to be set to a part of another string.
e.g. A$ = MID$(C$,3,4)

MID$ TO THE LEFT OF THE EQUALS SIGN.

This form will insert the string to the right of the equals sign into the substring specified by the MID$ command. The right side string to be inserted must have the same length as the MID$ command specifies. The MID$ string specified must already exist.

e.g. MID$(A$,1,3) = "yes"

In this, A$ must already exist and the "yes" length must be the same as the MID$ specifies, here 3.

MID$ ON BOTH SIDES OF THE EQUALS SIGN.

This form allows a part of one string to be inserted into another string. Both strings must exist before the instruction line is executed. The string length in each MID$ instruction must be the same.
e.g.

```
10 A$ = "AAAAAA"
20 B$ = "BBBBBB"
30 MID$(A$,3,2)=MID$(B$,1,2)
```

The above will produce A$= AABBAA

You can of course manipulate the same string in this way.
e.g.

```
10 A$ = "FRED BERT"
20 MID$(A$,1,4) = MID$(B$,6,4)
```

The example above will produce A$ = BERT BERT

Related Note: RIGHT$ AND LEFT$

These two commands do not exist in ZYBASIC; however this is no hardship as they can easily be simulated by the correct use of MID$. e.g.

LEFT$(str,exp) derived from MID$(str,1,exp)

RIGHT$(str,exp) derived from MID$(str,LEN(str)-exp,exp)

NEW NEW

Syntax:- NEW

Permits a new program to be entered.

(The keyword NEW can be replaced by "N.")

NEW is a direct mode command, i.e it cannot be used in a statement. It is used to start the entry of a new program file into the ZYBASIC text area. It resets all internal pointers and prepares the computer to receive text from either the keyboard or the tape interface. If you have accidentally typed NEW you can resurrect the original program provided you have not entered any new lines of text. See the OLD keyword for details.

The NEW command will force the following status to be set within ZYBASIC, (i.e. as though the following commands have been executed):

CLS	Clear the screen
DON	Dot markers on
SCROLL	VDU set to scroll mode
LINE 2	Scroll top is line 2

NEW does not affect the contents of any of the variables, (neither numeric nor string).

NEW is automatically executed as the first thing ZYBASIC does when it is set in operation by the user.

OLD OLD

Syntax:- OLD

This allows recovery of a ZYBASIC program which has been deleted by use of the NEW or BYE commands.

(The keyword "OLD" can be replaced by "O.")

OLD is a direct mode command and cannot be used in a statement line.

It will reinstate a previously entered or loaded program provided that no new lines are entered before the command OLD is issued.

To recover the old program after re-entry to ZYBASIC from ZYMON, type OLD followed by carriage return. OLD used in this way allows you to leave ZYBASIC and enter ZYMON without sacrificing the ZYBASIC program you had when you left. Once in ZYMON you could for example alter some machine code (probably used by ZYBASIC) and then return. To re-enter ZYBASIC use E A000 (C000 EPROM version) then OLD to continue with program development.

One less obvious use for OLD is during debugging, when the program running may have altered the screen mode inconveniently before the user breaks in. For example the screen may have been set to page mode near the bottom of the screen, displaying all results on only the bottom two lines: a simple way to restore the original display mode is to type N. carriage return, then O. carriage return

ON GOTO ON GOTO

Syntax:- ON var GOTO line,line,line,line
ON var GOTO var,var,line,var
(var,line,var, etc. in any combination)

ON GOTO is a size conditional branch command.

(The keyword "GOTO" can be replaced by "G."), e.g.

ON var G.line,line,var etc.

This command allows control to be passed to a line number dependent on the value of the variable preceding the GOTO. e.g.

```
10 ON Z GOTO 50, 100, 150, 180, 200
```

In this example control would be passed to lines 50, 100, 150, 180 or 200 depending on the integer value of the variable Z. If Z equals say 3.7, control is passed to line 150.

Note:- A useful feature of ZYBASIC is that a numeric variable can be used as a valid entry in the line number field after the GOTO. Control will be passed to the line number which equals the integer value of the variable.

```
5 SCROLL:LINE 1:DOFF:CLS
7 FOR N=1 TO 24
10 A = RND(5)
20 ON A GOTO 100,200,300,400,500
40 STOP
100 PRINT %1,A,"IS ONE"
110 GOTO 600
200 PRINT %1,A,"IS TWO"
210 GOTO 600
300 PRINT %1,A,"IS THREE"
310 GOTO 600
400 PRINT %1,A,"IS FOUR"
410 GOTO 600
500 PRINT %1,A,"IS FIVE"
600 FOR X=1 TO 5000:NEXT X   (Line 600 is merely a delay)
610 NEXT N
620 STOP
```

Notice the difference from the ON GOSUB command. After the ON GOTO command the program flow on completion of the subsequent branch does not have to return to the line after the ON GOTO command (as it must with ON GOSUB). Consequently it is free to continue to some other line number in some other part of the program.

ON GOSUB ON GOSUB

Syntax:- ON var GOSUB line,line,line,line

ON GOSUB is a size conditional call to a ZYBASIC subroutine.

(The keyword "GOSUB" can be replaced by "GOS."):
e.g. ON var GOS.line,line,line,var
(Line,line,var,etc. can be in any combination.)

This command allows control to be passed to a subroutine line depending on the value of the variable preceding the GOSUB.
e.g.

```
10 ON Z GOSUB 50, 100, 150, 180, 200
```

In this example a subroutine is called at one of lines 50, 100, 150, 180 or 200 depending on the integer value of the variable Z. If Z equals 3.2 say, or 3.95, a call is made to line 150.

Note:- In ZYBASIC, a numeric variable is allowed as a valid entry in the line number field after the GOSUB. The call will be to a line number according to the integer value of the variable.

```
10  SCROLL: LINE 1: DOFF: CLS
20  FOR N=1 TO 24
30  A = RND(5)
40  ON A GOSUB 100,200,300,400,500
50  FOR X=1 TO 5000:NEXT X
60  NEXT N
70  STOP
100 PRINT %1,A,"IS ONE"
110 RETURN
200 PRINT %1,A,"IS TWO"
210 RETURN
300 PRINT %1,A,"IS THREE"
310 RETURN
400 PRINT %1,A,"IS FOUR"
410 RETURN
500 PRINT %1,A,"IS FIVE"
510 RETURN
```

Notice the difference from the ON GOTO command. After ON GOSUB return has to be made to the line after that containing the ON GOSUB command, forcing the sequential execution of lines to be resumed.

OUT OUT

Syntax:- OUT port,data,data,data,data

Output to a Z80 port.

(The keyword "OUT" can be replaced by "O.")

"Port" is the Z80 port number and can be any var, or expression that resolves to an integer in the range 0 ~ 255 (#0 ~ #FF).

"Data" is any var, or expression that resolves to an integer value 0 ~ 255 (#0 ~ #FF)

Each additional data byte will be output to the next ascending port number

e.g.

```
10 OUT #30,1,6,56
```

Sends data 1 to port #30, data 6 to port #31 and data 56 to port #32

The following example assumes that a programmable sound generator (PSG) card is installed, with one of the PSG chips controlled by the ports #C0 and #C1.

```
1550 K=#C0
1560 L=#C1
1570 OUT K,7: OUT L,#FE
1580 OUT K,8: OUT L,3
1590 OUT K,0
1600 FOR X=1 TO 10
1610 FOR Y=1 TO 5000
1620 OUT L,Y
1630 NEXT Y
1640 NEXT X
1650 GOTO 1550
```

Note OUT port,var is standard BASIC, but OUT port,var,var is a ZYBASIC enhancement. Using this feature of ZYBASIC, lines 1570 and 1580 can be rewritten:

```
1550 K=#C0
1560 L=#C1
1570 OUT K,7,#FE
1580 OUT K,8,3
1590 OUT K,0
1600 FOR X=1 TO 10
1610 FOR Y=1 TO 5000
1620 OUT L,Y
1630 NEXT Y
1640 NEXT X
1650 GOTO 1550
```

You will be able to hear quite clearly the marked increase in speed of execution in the second version using the ZYBASIC-enhanced syntax.

PAGE PAGE

Syntax:- PAGE

PAGE sets the display mode of the VDU to be non scrolled.

(There is no abbreviation for the Keyword "PAGE")

Overflow from line 24 causes the next item to be displayed on the line specified by the LINE command.

Once in this mode of operation the LINE command specifies the line on which the next item to be printed on the VDU screen will appear. (The lines (or rows) of characters are counted from top to bottom, starting with line 1 as the topmost line, then line 2 below it and so on.)

By using the PAGE command in conjunction with the LINE command, an area of the VDU screen can be reserved for a continuous display (probably constructed with POKEs directly to the VDU RAM).

The following example shows how the screen could be divided into two areas to be used for different purposes:

```
10 CLS: DOFF
20 PAGE: LINE 1
30 PRINT TAB(10)"THIS IS A TITLE"
40 LINE 2
50 FOR A = 1 TO 100
60 PRINT A,"VARIABLE DATA AREA"
70 NEXT A
80 STOP
```

PEEK PEEK

Syntax:- PEEK(address)

This is a function and is returns the contents of the memory address that is within the brackets; the brackets are essential.

(The keyword "PEEK" can be replaced by "P.")

The (address) can be an expression, containing variables or other functions.

e.g.

```
10 A=PEEK(3+RND(6)/B)

10 IF PEEK(A+B)=32 GOTO 700
```

Note on #8000 (or -0 decimal)

The POKE command allows ZYBASIC to use values directly from memory locations in the machine store. The (address) parameter can be specified as a signed decimal integer, or directly in hexadecimal using the "#" notation (the examples below show this in use). Store addresses from #0000 to #7FFF correspond to 0 to +32767 in decimal, and #8001 to #FFFF correspond to -32767 to -1. There is however one address which cannot be used, namely #8000 (decimal 32768). The reason for this is that ZYBASIC stores numbers internally as "16-bit signed binary integers", and in this notation #8000 is meaningless as it would have to be interpreted as "-0". Since "0" is already quite adequately represented by #0000, there is no need for a representation for "-0" as well; it turns out more convenient to prohibit this number than to devise methods of coping with its use.

```
20 FOR A=#F000 TO #F2FF    VDU screen addresses
30 B=PEEK(A)               Read contents of screen
40 IF B<>32 GOTO 60        If the code is not a space jump
50 POKE A,42               Else make the space into a *
60 NEXT A                  Repeat for all screen

10 FOR X=1 TO 10           Do 10 times
20 FOR A=#F000 TO #F2FF    For the VDU screen
30 POKE A,PEEK(A)+1        Add one to each character
40 NEXT A                  Do it to whole screen
50 NEXT X                  Repeat until done
```

Line 20 in each of the above programs could be replaced by:

```
20 FOR A= -4096 TO -3329
```

since these numbers are the decimal equivalents of the VDU addresses #F000 to #F2FF; if you think this that signed decimal is an awkward way to express hex. addresses, you're right - think yourself lucky ZYBASIC lets you use hex.! (The use of decimal PEEKs and POKEs is one of the reasons magazine programs published for other computers are often so incomprehensible.)

To obtain the decimal equivalent of a hexadecimal number (say #F000) simply type P.#F000 in direct mode, and the result (-4096) will be printed.

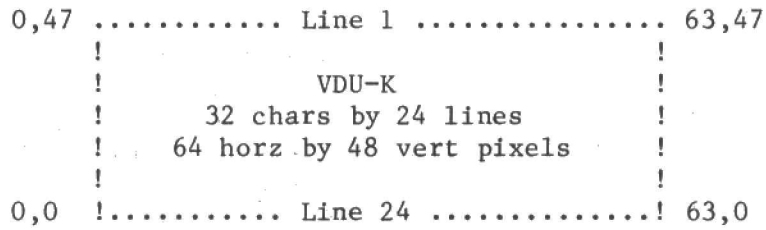
POINT POINT

Syntax:- POINT(horz,vert)

Point is a function and returns the pixel condition at the VDU-K screen location horz,vert.

(The keyword "POINT" can be replaced by "POI.")

The VDU-K screen has pixels such that there are 64 horizontal positions 0 to 63, and 48 vertical positions 0 to 47. Position 0,0 is the bottom left hand corner and 63,47 is the top right hand corner. (This differs from many BASICs which, somewhat illogically, adopt the top lefthand corner of the screen as the origin 0,0 - perhaps they were absent from school on the day they did graphs.)



On the VDU-K, X has a range from 0 thru 63 and Y has a range 0 thru 47. On the VDU-2K, X has a range from 0 thru 127 and Y has a range 0 thru 47.

If instead of a pixel at horz,vert there is a character, the ASCII code for that character is returned. Note that ASCII space qualifies as a character and returns 32 decimal. Only code zero returns pixel unset.

e.g. 20 A = POINT(12,34)

In the above example, if the pixel at position 12,34 is set (on), then the variable A is returned equal to -1. If the pixel is unset (off), A is returned equal to 0. If the letter C is at point 12,34 then A will be returned equal to 67 (since the ASCII code for the letter C is #43, and #43 expressed in decimal notation is 67).

```
10 A = POINT(12,34)          A = 0 if pixel unset.
                             A = -1 if the pixel is set.
                             A = ASCII if a character is present.

100 IF POINT(X,Y+3) = 0 GOTO 500


10 CLS:PAGE:DOFF:LINE 1
20 FOR X = 0 TO 63
30 FOR Y = 0 TO RND(48)-1
50 SET (X,Y)
60 NEXT Y
70 NEXT X
80 CLS:GOTO 20
```

POKE POKE

Syntax:- POKE address,data,data,data,data,data. . .

(The keyword "POKE" can be replaced by "PO.")

This command allows the programmer to write to main store. Each additional data item is written to the next incremental store address.

The "address" can be an expression containing variables or other functions.
e.g.

```
10 POKE 3+RND(6)/B,A

10 POKE #F1FF,35*45

10 POKE A+B,A-B
```

Note on #8000 (or -0 decimal)
The (address) parameter can be specified as a signed decimal integer, or directly in hexadecimal using the "#" notation (the examples below show this in use). Store addresses from #0000 to #7FFF correspond to 0 to +32767 in decimal, and #8001 to #FFFF correspond to -32767 to -1. There is however one address which cannot be used, namely #8000 (decimal 32768). The reason for this is that ZYBASIC stores numbers internally as "16-bit signed binary integers", and in this notation #8000 is meaningless as it would have to be interpreted as "-0". Since "0" is already quite adequately represented by #0000, there is no need for a representation for "-0" as well; it turns out more convenient to prohibit this number than to devise methods of coping with its use.

```
10 FOR A=#F000 TO #F2FF
20 FOR D=0 TO 255
30 POKE A,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D,D. . . D,D,D
40 NEXT D                        ↑ Use as many D's as you wish, (e.g. 32 to
50 NEXT A                          fill one full row of the VDU-K screen)

10 A=#F000
20 POKE A+RND(#2FF), RND(16)-1
30 GOTO 20

10 POKE #F100,#41,#42,#43,#44,#45  ;Display ABCDE on the VDU

10 POKE -3840,65,66,67,68,69       ;(Exactly the same result as above
                                    but using signed decimal instead of
                                    hex. for addresses and data.)
```

Note: It is a special feature of ZYBASIC to allow a single POKE command to be used with more than one item of data; in a conventional BASIC the last example would have had to be:

```
10 POKE -3840,65:POKE -3839,66:POKE -3838,67:POKE -3837,68:POKE -3836,69
```

PRINT PRINT

Syntax:- PRINT str
PRINT var
PRINT exp

(The keyword "PRINT" can be replaced by "P." An alternative abbreviation used by some other BASICs, and equally acceptable, is "?")

The string to be PRINTed must be put inside double quotation marks. This allows the single quote mark to be printed. If the end of the string coincides with the end of the line the closing quote may be omitted, as in line number 20 of the next example.

e.g.

```
10 PRINT "Yes I can, YES I CAN"
20 PRINT "No you can't, NO YOU CAN'T!
```

This will print the two statements on two lines. Note that carriage return cannot be used in a string as it will terminate the line. Items in separate print statements may be printed on the same line by putting a comma after the closing quote of the first print statement.

e.g.

```
10 PRINT "ARE YOU OVER 21? ",
20 PRINT "(REPLY Y OR N)
30 PRINT:PRINT:PRINT
```

Will print ARE YOU OVER 21? (REPLY Y OR N) followed by three carriage return line feeds. (i.e. three blank lines).

A variable may be printed with a string and must be separated from the string by a comma.

e.g.

```
10 P."TOTAL=",X,"TONS"
```

Here if X=20 then TOTAL= 20 TONS would be printed. ("PRINT" has been abbreviated to "P." in this example.) The symbol "?" is also valid as a PRINT command.

Numbers are printed with leading blanks and one trailing blank and take up a minimum total of 8 characters (spaces) in all. This may be altered by using the % sign and a number or expression.

e.g.

```
10 P."ANSWER IS",%3,H
```

Will print the value of H in three spaces if H is an integer number between -99 and 999.

(continued on next page)

If more digits are to be printed the number will be printed in the least number of spaces. The number of spaces specified for a number can be in the range 1 to 128:

e.g.

```
10 A=1: FOR N=1 TO 128: PRINT %N,A: NEXT A
```

e.g.

```
10 REM LOONEY NUMBERS
20 LINE 1: CLS: DOFF
30 FOR J = 0 TO 100
40 FOR K = 0 TO 20
50 PRINT %J,J,
60 NEXT K
70 NEXT J
80 GOTO 30
```

In a print statement a string variable can replace the items in the quotes.

e.g.

```
10 PRINT "FRED" is the same as:-
10 A$="FRED"
20 PRINT A$
```

Note: The behaviour of the PRINT command may be slightly modified in different versions of ZYBASIC, see Appendix E for some more information on this subject.

PROFF PROFF

Syntax:- PROFF

This keyword will stop printer 2 from echoing screen output.

(There is no abbreviation for "PROFF".)

PROFF can be used within a statement line or can be typed directly from the keyboard. It will stop data printed to the VDU being reflected to the printer. It can be used to strip unwanted text from a program's printed output.

```
10 PROFF
20 PRINT "THIS IS NOT PRINTED"
30 PRON
40 PRINT "THIS IS PRINTED"
50 PROFF
60 STOP
```

PROFF and PRON are used together to control print output. See the PRON command description.

Also see the entry in this manual describing TAB for some more information relating to the use of printers.

ZYBASIC expects the printer to be operated via two ports: Port 6 for the Status (Bit 7 to be active high as "TBMT", "transmit buffer empty", for a UART and "not busy" for a parallel interface) and Port 7 for the data. There are more remarks on hardware at the beginning of this manual.

ASCII codes are sent to the printer, and to help those building their own hardware, here is the source code of the ZYBASIC hardcopy driver routine, which is called when there is data in the A register ready to print:

```
TEST:   LD C,6          ;Status Port No. into Register C
        IN B,C          ;Read the Status into B
        BIT 7,B         ;Test Bit 7
        JR Z, TEST      ;Again if it's 0 (i.e. busy, not empty)
        INC C           ;Otherwise bump C to Data Port (6)
        OUT (C),A       ;Print the data
        CP 0D           ;See if it was #0D, i.e. carriage return
        LD A,0A         ;Load #0A, line feed in case it was
        JR Z,TEST       ;And maybe print line feed
                        ;Then proceed
```

PRON PRON

Syntax:- PRON

This command will cause all printable characters to be sent to printer 2 as they are sent to the VDU.

(There is no abbreviation for "PRON".)

PRON can be used within a statement line or it can be typed directly from the keyboard. It will cause data printed to the VDU to be reflected to the printer. It can be used to print required text from a program's printed output.

```
10 PROFF
20 PRINT "THIS IS NOT PRINTED"
30 PRON
40 PRINT "THIS IS PRINTED"
50 PROFF
60 STOP
```

PRON and PROFF are used one with the other to control print output. Also see the PROFF command description.

(There is some more information relating to the use of printers in the description of the TAB keyword.)

ZYBASIC expects the printer to be operated via two ports: Port 6 for the Status (Bit 7 to be active high as "TBMT", "transmit buffer empty", for a UART and "not busy" for a parallel interface) and Port 7 for the data. There are more remarks on hardware at the beginning of this manual.

READ READ

Syntax:- READ variable
READ string

(There is no abbreviation for "READ".)

When a READ command is found for the first time while executing a program, a search is made for the first DATA line in the program. The variable or variables following the READ command are then given the values of the numerical information found in the DATA line.

```
100 READ A,B,C
200 DATA 3.14159,2.8,88
```

Variables A,B and C will be assigned the values 3.14159, 2.8, 88 using the DATA line. When the next READ statement is reached, the next specified variable will be assigned the next available data value. An error appearing in a line containing a READ command may indicate an error in a line containing DATA, or that the DATA does not exist, or that all DATA has been READ.

Strings may be held in DATA statements and may be READ into string variables. The RESTORE command is used to restore the DATA pointer to enable the DATA to be READ more than once.

```
10  T1=0:  REM TAB TO NAME
20  T2=10: REM TAB TO AGE
30  RESTORE
40  CLS: DOFF: SCROLL: LINE 1
50  PRINT TAB(T1)"NAME",TAB(T2-1)"AGE"
60  READ N$
70  IF N$="END" PRINT: PRINT "END OF LIST": STOP
80  READ A
90  PRINT TAB(T1)N$,TAB(T2)%2,A
100 GOTO 20
110 DATA "Smith",30,"Brown",20
120 DATA "Parks",18,"Jones",55
130 DATA "Collins",13,"Doe",4
140 DATA "END"
```

When RUN, this program produces:

```
NAME     AGE
Smith     30
Brown     20
Parks     18
Jones     55
Collins   13
Doe        4
```

REM REM

Syntax:- REM any text

(The keyword "REM" can be replaced by "!")

This command instructs the computer to ignore the rest of the program line. This allows you to insert comments into your program for documentation. If REM is used in a multi-statement line, it must be the last statement.

```
100 REM ** THIS IS A REMARK AND WILL NOT BE EXECUTED  **
110 REM ** IT IS USED TO DESCRIBE A SUBROUTINE  OR TO **
120 REM **   GIVE THE PROGRAM A NAME.  IT CAN ALSO    **
130 REM **     BE USED TO INCLUDE SAY OPERATING       **
140 REM **       INSTRUCTIONS IN A LISTING.           **
150 PRINT "ONLY LINE 150 IS EXECUTED"
160 ! THIS IS ALSO A REMARK LINE BUT USES THE ABBREVIATION "!"
170 !   AS BASIC PROGRAMS ARE MOSTLY WRITTEN IN UPPER CASE,
180 !      Remarks may be more noticable in lower case.
```

```
10 REM  Remarks can be included on multi statement lines, e.g.
20 A=1: REM Set A to 1, next add one to it: A=A+1
30 REM  ** (But not after the RETURN keyword)
```

Any alphanumeric character which can be typed may be included in a REM statement. The control characters carriage return and Control-C however cannot be used as they will terminate the line.

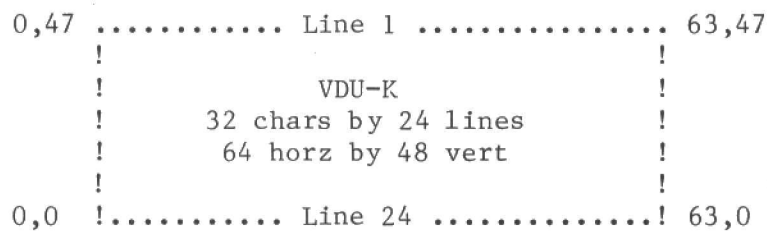
(See also the entry in this manual for the RETURN keyword, for some remarks on the use of remark statements within a program.)

RESET RESET

Syntax:- RESET(horz,vert)

(There is no abbreviation for "RESET".)

The VDU-K screen has pixels such that there are 64 horizontal positions 0 to 63, and 48 vertical positions 0 to 47. Position 0,0 is the bottom left hand corner and 63,47 is the top right hand corner. Many BASICs use the top left hand corner as 0,0 but as this offers no benefit to the user, ZYBASIC specifies the more logical bottom left hand corner as the 0,0 origin.



Reset pixel x,y. (VDU-K and VDU-2K only.)

On the VDU-K, X has a range of 0 thru 63 and Y has a range 0 thru 47. On the VDU-2K, X has a range 0 thru 127 and Y has a range 0 thru 47.

```
950  REM BOUNCING BALL
1000 CLS:DOFF:LINE 1:PAGE
1010 FOR M=1 TO 62
1020 SET(M,1):SET(M,46):NEXT M
1030 FOR M=1 TO 46
1040 SET(1,M):SET(62,M):NEXT M
1050 X=2:Y=2:X1=1:Y1=1
1060 X2=X+X1:Y2=Y+Y1
1070 IF POINT(X2,Y2)=-1 GOTO 1140
1080 RESET(X,Y):SET(X2,Y2)
1100 X=X2:Y=Y2:GOTO 1060
1140 IF (X2=1)+(X2=62) X1=-X1:GOTO 1060
1150 Y1=-Y1:GOTO 1060
```

RESTORE RESTORE

Syntax:- RESTORE

(The keyword "RESTORE" can be replaced by "RES.")

This command RESTOREs the DATA pointer, allowing DATA information to be read more than once during a program's execution.

e.g.

```
10 PAGE
20 LINE 1
30 CLS
40 DOFF
50 FOR A = 1 TO 2
60    RESTORE
70    FOR D = 1 TO 10
80      READ N
90      PRINT "PASS",%1,A,"DATA ITEM",%2,D,"IS",%2,N
100   NEXT D
110 NEXT A
120 DATA 1,2,3,4,5,6,7,8,9,10
```

(The extra spaces in lines 60 to 100 may be omitted; they are only included as an example of how a procedures within a program can be highlighted. Fans of "structured programming" (if they ever lower themselves to use BASIC at all) seem to go in for a lot of this kind of thing. We mustn't however poke too much fun at them because they are probably right about the advantages of their method over the regular "non-structured sphagetti-multi-GOTO programming" which the rest of us use.)

Typing RUN to begin program execution in ZYBASIC will cause the DATA pointer to be set to the first DATA item. Although it is therefore not essential to RESTORE at the beginning of a program, good programing practice is to use a RESTORE statement on entry to any program which has DATA and READ statements.

```
10 CLS:      REM * Clear the screen
11 PAGE:     REM * Set page mode
12 LINE 1:   REM * Set line 1 to start
13 DOFF:     REM * Turn off the dots
14 RESTORE:  REM * RESET THE DATA POINTER
20 READ X:   REM * Get the first data item as X
30 RESTORE:  REM * RESET THE DATA POINTER
40 READ Y:   REM * Get the first data item as Y
50 SET(X,Y):REM * Set pixel X,Y
60 STOP:     REM * End
70 DATA 15: REM * Data for program use
```

RETURN RETURN

Syntax:- RETURN

(The keyword "RETURN" can be replaced by "RET.")

This is used in conjunction with GOSUB, to allow the user to write a "subroutine" (i.e. an often-used sequence of statements, such as a delay subroutine). The subroutine is called by the GOSUB; i.e. control passes to the subroutine until the RETURN at the end is executed - then control returns to the main program at the point it left off. The outstanding benefit a subroutine offers is that it can be called from anywhere in the program, whenever it is needed. Without the subroutine a whole sequence would have to be written out again in full each time it was needed.

```
10 PAGE:LINE 1:DOFF:CLS
20 FOR A=1 TO 3
30 FOR X=0 TO 23
40 PRINT X,
50 NEXT X
60 GOSUB 1000
70 NEXT A
80 PRINT "RUN ENDS"
90 GOSUB 1000
100 GOTO 10
997 !    * * * * * * * * * * *
998 REM  * DELAY SUBROUTINE  *
999 !    * * * * * * * * * * *
1000 FOR D=0 TO 1500
1010 NEXT D
1020 RETURN
```

Note a useful tip from the above program: When you write a subroutine it is best to name it, in case you forget its purpose, or simply to aid study later. To name something so that it stands out in a listing, use three REMs in a group with the outer two highlighted, e.g. with asterisks as in line numbers 997 and 999 above.

The REM statements add slightly to the time it takes for the program to run, because ZYBASIC still has to interpret a REM just to find out it should be ignored. You may want to delete REMs is if you are very short of memory: each character removed in a REM statement is an extra character you can use for program or array storage. Some people often keep two versions of a program; one with the REMs to be studied, and one without the REMs to run. To make it easy to delete REMs in a program never call a subroutine via a REM line. It is good programming "manners" (helpful if you are writing a program for others to see) to have subroutines begin in nice round numbers, e.g. 1000, 2200, 5000, etc., with the REM statements naming them immediately before, e.g. 999, 2199, 4999, etc.

Note that a REM statement cannot follow a RETURN. An error message will be produced if you write e.g. 1020 RETURN: REM Exit to main loop.

RND RND

Syntax:- RND(num)

(The keyword "RND" can be replaced by "R.")

RND(X) gives a random number between zero and one if the value of the expression in brackets is zero, or a number between one and X if the value is greater than zero.

```
10 A = RND(0)     Produces a 6 digit random fraction.
```

```
10 A = RND(45)    Produces a random number in the range 1-45
```

```
10  REM **  DISTRIBUTION OF RND(64)-1  **
20  REM ** ~~~~~~~~~~~~~~~~~~~~~~~~~~ **
30  REM ** THE HEIGHT IS THE NUMBER OF **
40  REM **   TIMES RND(64)-1 IS THE    **
50  REM **   X AXIS POSITION 0 TO 63   **
60  REM ** ~~~~~~~~~~~~~~~~~~~~~~~~~~ **
70  PAGE: CLS: DOFF: LINE 1
80  X=RND(64)-1
90  Y=0
100 Z=POINT(X,Y)
110 IF Z<>0 GOTO 1000
120 SET(X,Y)
130 GOTO 80
1000 IF Y<47 Y=Y+1:GOTO 100
1010 STOP
```

The above program produces a histogram of the distribution frequency of the random number algorithm that is built into the ZYBASIC interpreter. A random number between 0 and 63 is produced and plotted to the next free Y position along the X axis. This generates a diagram showing the number of times each X position has been selected. The theoretically perfect random generator would produce a flat topped graph. The program will run forever and produce a more and more accurate graph, except that it will eventually run out of screen space.

To study the random number process further you could alter the program to count the occurrences and store the results in an array, finally displaying the relative differences. If you do it please pass it to the user group newsletter, with the results of your test, as it would be most interesting to see the overall "goodness" of this algorithm fully analysed.

RUN RUN

Syntax:- RUN

This is a direct mode command and cannot be used in a statement line.

(The keyword "RUN" can be replaced by "R.")

RUN is used to start of a ZYBASIC program that has been stored in the text area of the computer's memory.

Typing RUN followed by a carriage return causes the computer to begin execution of the program starting at the lowest line number. At this point all variables are reset to zero and all strings are empty and unassigned. ZYBASIC's single array is staticised to contain all of the store space which does not contain statement lines. (As each array element is four bytes long the array size at RUN time is the spare memory space divided by 4).

GOTO line number carriage return, may be used instead of RUN. If GOTO is used the variables will not be cleared to zero and the strings will remain whatever they were before. This is a useful way of passing parameters to a program for processing whilst in the direct or calculator mode of operation. e.g. Enter the following program:-

```
5 PRINT A,B
10 PRINT A*B
20 PRINT A/B
30 PRINT A-B
40 PRINT A$
```

Now type the following. (Type carriage return after each line).

```
A=56
B=5
A$="SOME STRING"
GOTO 5
```

Compare this with what happens when you type RUN.

SAVE SAVE

Syntax:- SAVE line,numb

This is a direct mode command; it cannot be used in a statement line.

(The keyword "SAVE" can be replaced by "S.")

SAVE allows programs or parts of programs to be saved on tape and has works in a similar way to the LIST command. To use it start the tape machine on record, type carriage return and finally type SAVE carriage return. The complete program will be saved and the prompt will be printed when finished. ZYBASIC gives reassurance that something is happening during a SAVE by listing everything to the screen as well. To save a small section of program use the same format as the LIST command e.g. SAVE 100,1 will save line 100. Lines saved in this way are loaded simply by rewinding and replaying the tape. The lines will be inserted in the existing program. If a complete program is to be loaded rather than parts to be added to an existing one, NEW carriage return must be typed before replaying the tape.

MERGING OF PROGRAMS: It is possible to load a second program from cassette into memory without affecting the first program. The line numbers of the second program must not lie within the line number limits of the first program. Once this is done the first program can use parts of the second program for GOSUB calls; or the second program may be used independently using the GOTO command, instead of the RUN command, for program execution.

The ZYBASIC tape cassette routine was designed to run at 300 or 600 baud. The software will however work at 1200 or even 2400 baud but you must bear the following in mind:-

A program that is loaded from tape, and which has line numbers that match a program that already exists in the store, will overwrite the old lines with the new lines, leaving any other lines untouched. If the new lines are different in length from the original then the ZYBASIC will have to rework the program store to accommodate the new lines. As this takes an appreciable time then, at very high baud rates, data may be missed.

The way to ensure a good load at baud rates greater than 600 is to use the NEW command before loading the program from tape. This empties the text area and the new program will load with the line numbers in sequence without the problem of fitting new lines in an old program.

An unintended but nevertheless convenient use for the SAVE command is for listing programs on the screen. Because of the very high speed of screen access the LIST command is so fast that it is impossible to read a long program if it is listed in full. By using SAVE instead, the screen display will be slowed to the speed to which the tape interface is set: e.g. 2400 baud (still quite fast), 1200 baud (slower), or 300 baud (slow enough to read comfortably).

SCROLL SCROLL

Syntax:- SCROLL

(The Keyword "SCROLL" can be replaced by "SC.")

This command causes the VDU to enter the scrolling mode of operation. In this mode printed data is output to the VDU on line 24. Reaching the end of line 24, or typing a carriage return will cause the screen display to scroll, i.e. everything entered so far will move up one line to leave line 24 clear for more. (This is like the operation of a typewriter, where the paper moves up for each new line: the PAGE mode (q.v.) is like handwriting - the paper mostly stays still and the hand moves down for each new line.)

It is important to note that the screen will scroll from line 24 to the line specified in the LINE command. e.g.

If the following sequence was executed:-

```
10 CLS
20 SCROLL
30 LINE 10
```

Any scroll will now encompass lines 24 through 10 only. Lines 9 through 1 will not be affected. The use of this is obvious: a scrolling window can be defined with a fixed area left above that window.

When using the scroll mode, the line command will generate an error if LINE 24 is commanded. This is because you cannot scroll 24 to 24! The greatest line number that you can specify whilst in SCROLL mode, is LINE 23.

Try this program:

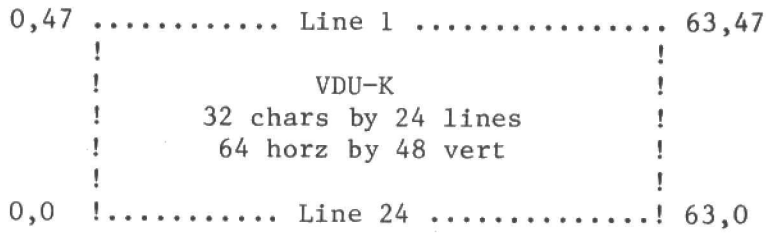
```
10 CLS: DOFF: PAGE: LINE 1
20 FOR A = 1 TO 15
30 PRINT "THIS AREA IS FIXED"
40 NEXT A
50 SCROLL: LINE 15
60 FOR A = 1 TO 900
70 PRINT %3,A,"ONLY USING THE BOTTOM HALF"
80 NEXT A
90 LINE 1
```

SET SET

Syntax:- SET(horz,vert)

(There is no abbreviation for the word "SET".)

The VDU-K screen has pixels such that there are 64 horizontal positions 0 to 63, and 48 vertical positions 0 to 47. Position 0,0 is the bottom left hand corner and 63,47 is the top right hand corner.



(Set pixel x,y cannot be used on the Kemitron VDU-A,B,G; the pixels here are the block graphics from the VDU-K and VDU-2K.)

On the VDU-K, X has a range of 0 through 63 and Y has a range 0 through 47. On the VDU-2K, X has a range 0 through 127 and Y has a range 0 through 47.

```
10 PAGE:LINE 1:DOFF:CLS
20 FOR X=0 TO 63
30 FOR Y=0 TO RND(46)
40 SET (X,Y)
50 NEXT Y
60 NEXT X
```

```
57 REM ****************************
58 REM *  SIN, COS, SIN+COS WAVES  *
59 REM ****************************
60 W=360/63:  REM HORIZONTAL SCALING FACTOR
70 A=16:      REM VERTICAL AMPLITUDE
80 CLS: PAGE: DOFF: LINE 1
90 FOR X=1 TO 63
100 SET (X,23+(A*SIN(X*W)))
110 SET (X,23+(A*COS(X*W)))
120 SET (X,23+(A*SIN(X*W)+(A*COS(X*W))))
130 NEXT X
```

```
200 CLS:PAGE:LINE 1:DOFF
210 SET(RND(64)-1,RND(48)-1)
220 GOTO 210
```

SIN SIN

Syntax:- SIN(num)
SIN(var)
SIN(exp)

It is a function and may be embedded within expressions.

(There is no abbreviation for the keyword "SIN".)

Returns the sine of an angle that was expressed in degrees.

```
10 A = SIN(30)          A is the sine of 30 degrees

10 A = SIN(30)+SIN(60)  A is set to the sine of 30 degrees plus the
                        sine of 60 degrees

10 PRINT SIN(A)         This will print the sine of the variable A
                        where A is expressed in degrees

10 Y = SIN(X+3.3)       Produces the sine of the sum of the two
                        degrees, X and 3.3

10 IF SIN(A+X) = SIN(A-X) GOTO 50
                        This uses the SIN function in a conditional
                        branch to decide the outcome of some
                        calculation.
```

Try these examples:-

```
1  REM PRINT A TABLE OF SINES
10 CLS:DOFF:PAGE:LINE 1            Initialise
50 PRINT "Angle....Sine"           Print a title
60   FOR X=10 TO 180 STEP 10       X begins at 10
70     PRINT %4,X,,,,,%1,SIN(X)    Print angle and its sine
80   NEXT X                        Loop if X not 190
90 STOP                            End
```

```
1  REM PRINT A SINE WAVE
10 CLS:DOFF:PAGE:LINE1             Initialise
20   FOR X=1 TO 63                 For X begins at 1
30     Y=24+(20*SIN(X*10))         Produce sine wave point
40     SET (X,Y)                   Display graphic point
50   NEXT X                        Loop if X not 64
60 STOP                            End
```

Line 30 in the last example could be rewritten SET(X,24+(20*SIN(X*10))) - this would eliminate the need for line 40 and so speed up the display.

Angles greater than 360 degrees will be computed with reducing precision as the total number of degrees becomes large with respect to the first circle of 360 degrees.

SINR SINR

Syntax:- SINR(num)
SINR(var)
SINR(exp)

It is a function and may be embedded within expressions.

(The keyword "SINR" can be replaced by "SI.")

Returns the sine of an angle that was expressed in radians.

```
10 A = SINR(3)          A is the sine of 3 radians

10 A = SINR(3)+SINR(6)  A is set to the sine of 3 radians plus the
                        sine of 6 radians

10 PRINT SINR(A)        This will print the sine of the variable A
                        where A is expressed in radians.

10 Y = SINR(X+3.3)      Producing the sine of the sum of the two
                        radians, X and 3.3

10 IF SINR(A+X) = SINR(A-X) GOTO 50
                        This uses the SINR function in a
                        conditional branch to decide the outcome of
                        some calculation.
```

Notes:-
A complete circle of 360 degrees is expressed as two pi radians where pi is 3.14159. So if P = 3.14159 = pi then:

```
 90 degrees = 0.5 * P = 1.57079 rads or pi/2 rads
180 degrees = 1.0 * P = 3.14159 rads or pi rads
270 degrees = 1.5 * P = 4.71239 rads or pi*(3/2) rads
360 degrees = 2.0 * P = 6.28319 rads or pi*2 rads
```

As 360 degs = 6.28319 radians, 1 radian = 360/6.28319 (i.e. 1 radian = 57.29574 degrees).

Also 1 degree = 6.28319 rads/360 = 0.01745 (i.e. 1 degree = 0.01745 radians).

Angles greater than 6.28319 radians will be computed with reducing precision as the total number of radians becomes large with respect to the first circle of 2 pi radians.

STOP STOP

Syntax:- STOP

(The keyword "STOP" can be replaced by "S.")

This command will terminate a program. It is not compulsory, as a program automatically terminates if there are no more line numbers to be executed. It is useful as a debugging aid as it may be inserted temporarily to halt a program part of the way through execution. Also STOP is useful in preventing the main loop of a program from running into a series of subroutines placed at the end of that program.

```
10 PRINT "This statement is
20 PRINT "before the
30 PRINT "STOP statement
40 PRINT:PRINT:PRINT
50 STOP
60 PRINT "You won't see this
70 PRINT "unless the STOP statement
80 PRINT "at line 50 is removed
86 REM
87 REM    (You can have more than
88 REM     one STOP in a program)
89 REM       e.g. line 90 next
90 STOP
```

SQR SQR

Syntax:- SQR(num)
SQR(var)
SQR(expression)

This is a function and it returns the square root of the expression in the brackets.

(The keyword "SQR" can be replaced by "S.")

```
10 A=SQR(4)      A is set to the square root of 4  (i.e. A=2)

10 A=SQR(34*SIN(30)+5)

10 PRINT SQR(5.6)

10 FOR X=1 TO SQR(16)
```

Try this program:-

```
10 CLS: DOFF: LINE 1: PAGE
20 FOR X=1 TO 63
30 Y=SQR(X)*5
40 SET (X,Y)
50 NEXT X
60 STOP
```

The program could be made shorter:-

```
10 CLS: DOFF: LINE 1: PAGE
20 FOR X=1 TO 63
30 SET(X,SQR(X)*5)
40 NEXT X
50 STOP
```

Or it could be written using multiple-statement lines as:-

```
10 CLS: DOFF: LINE 1: PAGE
20 FOR X=1 TO 63 : SET(X,SQR(X)*5) : NEXT X
30 STOP
```

Mathematicians will not need to be told that the square root of a negative number cannot be expressed in any of ZYBASIC's normal number systems, and so this operation is prohibited.

TAB TAB

Syntax:- TAB(num)

This prints spaces until the print head (both the real one if a printer is in use, and the imaginary equivalent which prints on the VDU screen) reaches the specified position on the current line. If the print head is already past the position then no action is taken.

(The keyword "TAB" can be replaced by "T.")

The maximum tab position is 255, which will appear on the VDU-K screen as several lines down, but if the output is given to a printer via PRON, then the TAB will correctly position the printer head mechanism.

The first position on the line is 1. (i.e. there is no position 0.)

```
10 PRINT TAB(10),3.145
```

```
10 PRINT TAB(A)
```

Try this program

```
10 CLS:DOFF:LINE 1:PAGE
20 FOR X=1 TO 20
30 PRINT TAB(X),"1"
40 NEXT X
```

The purpose of TAB is primarily for control of a printer, to enable quite complex hardcopy printouts to be produced without suffering any restrictions due to the limitations of space on the VDU-K screen. To suit this purpose, TAB issues ASCII space characters (#20) to position the print head. As shown in the examples above TAB does have some value in altering the print position on the VDU screen, but note that its use for this secondary purpose is somewhat limited because when the TAB command takes effect the ASCII "space" characters will overwrite (i.e. erase) any text etc. which is is already on the screen in their path.

USR USR

Syntax:- USR num
USR var
USR exp

(The keyword "USR" can be replaced by "U.")

USR and CALL are synonymous and enable a machine code subroutine to be called from within a ZYBASIC program. ZYBASIC responds to both keywords to improve program portability: USR = CALL = USR; use the one you prefer.

The computer will execute code from the indicated address until a return (hex. C9) is encountered; execution will then return to ZYBASIC at the point it left off.

So that the return will be successful, and so that ZYBASIC will be in the same state as it was before the USR, it is vital that ZYBASIC's stack area is not altered by the user routine. (Some user routines will not need to use a stack, but if yours does the recommended way to avoid disturbing ZYBASIC's stack is to leave it strictly alone and to use a different one for your subroutine.)

Once you change to a different stack you can no longer use the simple RET (hex. C9) machine-code instruction to get back. In these circumstances another method can be used to return control to ZYBASIC at the end of the USR machine code subroutine. The method which follows is based on the fact that immediately after the ZYBASIC USR command is executed, (i.e on entry to a machine code subroutine), the BC register pair contains the required return address:

```
10 USR #800     Start executing machine code from #0800
            !     (ZYBASIC will remember where its own stack was
            !        and will reload it at the return)
            !
      0800  LD SP,0FFFH              Log in to a new stack
            PUSH BC                  Save return address
            ......
            Move the space invaders      (Example user
            Bleep the sound card            routine)
            Update deadmen counters
            ......
            !
            POP HL                   Get return address
            JP (HL)                  Return to ZYBASIC
```

If the user's routine does not use the BC register pair, BC will still hold the return address; there is no need for the PUSH BC instruction above and the return to ZYBASIC can be made as follows:

```
LD HL,0
ADD HL,BC
JP (HL)
```

(Description continues on next page)

USR (continued from previous page) USR

The following store areas are not used by ZYBASIC, and can be used for your own machine code programs, data, etc.:-

```
0800 ~ 0FFF hex. ...not used by ZYBASIC
9B00 ~ 9FFF hex. ...not used by ZYBASIC
```

You may also use some of the space 1800 ~ 7FFF hex. but since this is the area where the program text is stored the amount of space available here depends on the size of your ZYBASIC program. Unless the ZYBASIC program is very big there will generally be a few K spare at the top of the text area. (You can find out by using the FREE command after loading the ZYBASIC program.) Note that the USR subroutine must not unintentionally alter ZYBASIC or the program text area. If it does, hard to predict results would be obtained on the return to the corrupted ZYBASIC and its program.

```
      0000 - 07FF  ZYMON
free..0800 - 0FFF  (Top area is ZYMON's stack etc.)
      1000 - 1800  ZYBASIC numeric variables, stack and scratch pad
      1800 - 7FFF  ZYBASIC program text area, user code
      8000 - 9AFF  ZYBASIC string variables
free..9B00 - 9FFF
      A000 - BFFF  ZYBASIC interpreter
```

DERIVED COMMANDS DERIVED COMMANDS

The following two common BASIC commands do not exist in ZYBASIC but can be easily derived from keywords that do exist.

LEFT$(str,exp) can be derived from MID$(str,1,exp)

RIGHT$(str,exp) can be derived from MID$(str,LEN(str)-exp,exp)

(The above expressions are repeated in this manual under the headings MID$ and LEN$, as a reminder that LEFT$ and RIGHT$ can be derived if need be.)

(This page intentionally left blank so as to begin next section on a new page)

APPENDIX A APPENDIX A

4.1 Appendix A

ABBREVIATIONS AND COMMAND GROUPS

KEYWORD	ABBREVIATION	GROUP
LIST	.NL	Direct
RUN	R.NL	Direct
NEW	N.NL	Direct
BYE	B.NL	direct
SAVE	S.NL	Direct
OLD	O.NL	Direct
AUTO	A.NL	Direct
ED		Direct
SET(x,y)		Graphic
RESET(x,y)		Graphic
POINT(x,y)	POI.(x,y)	Graphic
FREE	F.	Function
LEN($)		Function
RND(x)	R.(x)	Function
ABS(x)	A.(x)	Function
SQR(x)	S.(x)	Function
INT(x)	I.(x)	Function
PEEK(x)	P.(x)	Function
IN(x)		Function
SINR(x)	SI.(x)	Function
SIN(x)		Function
COSR(x)	C.(x)	Function
COS(x)		Function
CLS		Control
LINE n		Control
DON		Controol
DOFF		Control
PAGE	PA.	Control
SCROLL	SC.	Control
TAB(x)	T.(x)	Control
REM	!	Control
STOP	S.	Control

KEYWORD	ABBREVIATION	GROUP
INPUT$		String
MID$		String
LET$	default	String
LET	default	Assignment
FOR	F.	Branch
TO	TO	Branch
STEP	S.	Branch
NEXT	N.	Branch
GOTO	G.	Branch
GOSUB	GOS.	Subroutine
RETURN	RET.	Subroutine
CALL	CA.	Subroutine
USR	U.	Subroutine
PRINT	P.or ?	Output
POKE	PO.	Output
OUT	O.	Output
@	@	Output
INPUT	IN.	Input
READ	READ	Input
DATA	D.	Input
RESTORE	RES.	Input
INKEY	INK.	Input
@	@	Input
IF	IF	Conditional
ON	ON	Conditional

Programs written using the abbreviated form of the commmands occupy less store space and run faster. They are generally harder to understand, but this makes them less attractive to steal, so this can be an advantage!

4.2 Appendix B

FURTHER NOTES ON STRINGS

A$ to Z$ are the 26 string variables. Although the rest of ZYBASIC 2 will work without RAM at location #8000, you can't have any string variables unless you put some RAM there. (If ZYBASIC 2A is being used the necessary RAM will almost certainly be present since there is bound to be RAM around this area for ZYBASIC itself.) ZYBASIC 2C runs in EPROM and so it is less certain that RAM will be present at #8000. You can put a little or a lot of RAM starting at 8000H; for each 1K of RAM you provide you will get another four string variables to use. The first 1K will let you use A$, B$, C$, and D$. The next 1K you add will let you use the next four, i.e. E$, F$, G$, and H$, so if you want all twenty-six then you will have to provide six and a half K. Anyone who is using the RRM-14 card for this RAM will be a teeny-weeny bit miserable, because they will have to make do with just twenty-four string variables since they only have 6K of RAM available on the card and 6 times 4 = 24. (If you don't believe me type this into your computer, running ZYBASIC 2: PRINT 6 * 4). In short, RRM-14 users - no Y$ or Z$ for you. Most ZYBASIC users won't be using the Static RAM on the RRM-14, as the use of the Dynamic RAM provides a cost-effective financial saving syndrome compared with Static RAM, i.e. per K it's cheaper.

Advertisement: A "Great Idea", putting the strings in a fixed location like this! Why? Because they use RAM separate to that allocated for the program text area. Nobody ever seemed to visit the desolate wastes of #8000 much before, it was all a bit arid until we planted the RAM there. Also this way the strings don't move - if they weren't in these fixed locations and had to bash about in the ordinary RAM, competing with everything else (the other variables, the BASIC text store, the array, the scratch pad, and so on), then it would be hard for you to find them, if you wanted to PEEK and POKE them. Why do you want to do that anyway? Well because, didn't you notice, these strings are dirty great strings (I thought that too when I first saw them, but was too shy to say anything). So dirty and so great in fact that just three of them will pretty well fill a VDU screen, if they were to be PRINT-ed to it. And boy is that fast, and boy, if you've got a VDU-K, with all those dinky little programmed characters, and no snow, then you can compose some quite good fast-moving animated displays, limited only by the programmer's imagination. (Of course if you've got a really good programmer's imagination you don't even have to switch the computer on - all you have to do is imagine you've done it.)

More advertisement: (Patience, patience, my friend, I know you're only reading this to find out how to use the strings, well you'll just have to wait until after the commercial is over!) Another great benefit of having the strings how they are and where they are is that they can soak up large quantities of executable machine code. If your BASIC program needs some bits (or bytes, I don't care) of machine code, then you can cart them around tied up in a length of string. For instance A$ doesn't have to be a string of ASCII characters for you to read, it can be a string of bytes which represent an executable machine code program of

length approaching a quarter of a K. In some circumstances this can be a useful alternative to the normal method of POKE-ing into an area of RAM the code contained in a DATA statement. (Don't ask me what circumstances exactly, I really only said it so that I would sound as if I knew what I was talking about!)

At last! Here is how to use them. I'm sorry I haven't said what strings are yet, and what use they are apart from their inestimable value in being a source of never ending jokes (e.g. It's a "knotty" problem "unravelling" a "string" command if you start from the wrong "end", and other screamingly funny items of that ilk).

Some Examples:

A$ to Z$ exist (given sufficient memory above 8000H)

In Print statements you can use a string variable where you would use quotes ("), or mix them both, as if each was in quotes. The comma will introduce a space, the + will concatenate (string together).

```
10  LET  A$ = "ABCD"  (The final " may be omitted, as in the PRINT
                       statement and the LET may be omitted as well, as in
                       line 20 below)
20 B$ = "1234
30 PRINT A$+B$  result ABCD1234      (No Spaces)
40 PRINT A$,B$  result ABCD 1234     (Two strings with a space)
50 PRINT A$+"EFGH"+B$+"5678"
                       result ABCDEFGH12345678  (Like one big new string)
60 PRINT A$+"EFGH",BS+"5678"
                       result ABCDEFGH 12345678 (Two strings with a space)
```

In LET statements this is legal:

```
A$=B$+C$+"FRED"
```

But this is illegal:

```
A$=B$,C$,"FRED"
```

i.e. getting technical about it, a new string (A$ in this example) can be the sum of other strings, but it is illegal to redefine the same string more than once - it has to be one of the three, or the sum - schizophrenia would set in if it had to be all three at once.

LEN(A$)

This returns the numeric count of the length of the string. e.g. If the statement X=LEN(A$) is used on a line at the end of the program below (with a line number of course, if you're going to try it out), then X will end up equal to 8, the combined length of BILL+FRED.

```
10 A$="FRED"
20 B$="BILL"
30 X=LEN(A$+B$)
```

LEN (no relation to BILL or FRED) is used in this type of program:

```
Get String from some source (no jokes please)

FOR J=1 TO LEN(String)

Do Process

NEXT J
```

Note: Since LEN(String) returns a number this cannot be assigned to a string variable:

i.e A=LEN(B$) is legal because A is an ordinary variable, whereas A$=LEN(B$) is illegal (because A$ is a string variable).

INPUT$

This gets a string from the keyboard.

e.g. 10 A$=INPUT$

or 10 B$="CAPTAIN " (Note there is a space after the word CAPTAIN, in this example)
20 A$=B$+INPUT$

When this two line program is run it produces

```
CAPTAIN
```

and waits for input. If you input the words JAMES KIRK then A$ will end up equalling CAPTAIN JAMES KIRK.

If you really wanted to use this program then line 10 could be eliminated by including the B$ component in quotes i.e.

```
20 A$="CAPTAIN "+INPUT$
```

MID$ to the Right of the Equals Sign

If for example MID$(A$,3,2) is used to the right of the equals sign, it obtains the substring of the (string, start position, no. of characters) e.g.

```
10 A$="ABCDEFG"
20 B$=MID$(A$,3,2)
```

B$ will be set to CD, i.e. 2 characters have been extracted from ABCDEFG, starting with the 3rd.

Looking more closely at the syntax, the A$ in the above example can be any of the string variables B$, C$, D$, etc. The start position (3 in the above example) and number of characters (2 in the example) can in fact be functions of any kind e.g. any of the variables A-Z9, or RND( ) etc., e.g. LEN( ) or even RND(LEN( )) if you wanted.

The same rules apply as to LET, e.g.

```
20 B$=MID$((A$,3,2)+MID$(A$,5,2)
```

used in place of the line 20 in the last example will cause B$ to be set to CDEF, because CD are the 2 characters starting at position 3, and EF are the 2 characters starting at position 5.

MID$ to the Left of the Equals Sign

In this case whatever is to the right of the equals sign will be inserted into the string referenced on the left (subject to some rules not being broken). e.g.

```
10 A$="123456789    (A$ isn't part of the example, it's just to
                    help you count the dots in the next line)
20 B$="........."
30 MID$(B$,2,3)="BEN
```

After this B$ will be .BEN..... (the ",2" specified position 2, and the ",3" specified a total of 3 characters).

Note that the string B$ (".........") had to exist before the substring "BEN" could be inserted into it, and note also that the insertion length had to equal the substring length. For example line 30 MID$(B$,2,3)="BENNY would not be legal until the ",3" was changed to ",5" to match the 5 character length of the substring "BENNY":

```
30 MID$(B$,2,5)="BENNY.
```

When carrying out this kind of activity, where it is important to get the length exactly right, the previously described function LEN(String) will be very useful. e.g.

```
10 B$="1234567
20 C$=INPUT$              (We don't know what length C$ will be)
30 L=LEN(C$)              (We do now - it's L)
40 MID$(B$,1,L)=C$
```

(Lines 30 and 40 could be replaced by a single line MID$(B$,1,LEN(C$))=C$ in real life, once you knew what you were doing.)

Mind you, you have to do more than just get the length right. Another rule which has to be obeyed is that C$ must not exceed B$ in length, as you can't fit a cuckoo another bird's nest, where C$ stands for the cuckoo, and B$ for the bird's nest.

In case you can't work out how to make a test to ensure this rule isn't broken, you could try something of the form below:

```
IF LEN(C$)-LEN(B$)>0 PRINT "THAT'S TOO LONG YOU TWIT"
```

(If LEN(C$) is bigger than LEN(B$) then the result of the subtraction will indeed be greater than 0, and the rude message will be appropriate.)

MID$ on both sides of an equals sign

Suppose A$ = ".....", and B$ = "1234", and a line such as the one which follows is executed:

```
MID$(A$,2,3) = MID$(B$,1,3)
```

(Note the substring length, 3 in this example, has to be identical in both expressions)

Then the result will be that A$ becomes .123....

A few more things about strings

Remember a page or two ago? (Under the heading INPUT$). The example used was:

```
10 B$="CAPTAIN "
20 A$=B$+INPUT$
```

If you input the words JAMES KIRK at line 20 then A$ ends up equalling CAPTAIN JAMES KIRK.

Now just suppose CAPTAIN JAMES KIRK had been a naughty boy, and he had to be demoted for splitting his infinitives ("to boldly go"), then we could use the following line:

```
30 A$ = MID$(A$,9,10)
```

Result, A$ is now plain JAMES KIRK. By an amazing coincidence the "10" in the expression above just happened to match exactly the number of characters in the string "JAMES KIRK". The slight drawback to this technique is that you can only demote people who have names 10 characters long, including spaces.

Do you want to know how to cope? The answer is to use a slightly more complicated version of the proposed line 30:

```
30 A$ = MID$(A$,LEN(B$)+1,LEN(A$)-LEN(B$))
```

Now on to a new subject.

What is a useful little word which can be used with strings? No, not "please", (although a little politeness when addressing a computer is always a good idea if you want to get the best out of it), no, the word is "IF", a word made famous by Rudyard Kipling.

Unfortunately, the present state of development of ZYBASIC so far is such that it is only possible in string statements to use IF with equals (=) or not equals (<>), i.e you can't use less than (<) or greater than (>).

For example you can use IF as follows:

```
     IF A$ = B$ GOTO wherever-you-like
or
     IF A$ = "YES" . . .
or
     IF MID$(A$,LEN(A$)-2,3) = "YES"
```

Actually there is more to the last example than first meets the eye. The string commands LEFT$ and RIGHT$ are absent from the present version of ZYBASIC, but there are ways of managing without. A closer look at the last example should show that it is a way of simulating the RIGHT$ command; if you don't know what RIGHT$ is supposed to do anyway then why worry - you'll never miss what you never had! (There's a right way to do things, even if there isn't a RIGHT$ way to do them.)

For example, if you wanted the right three characters from the string "ABCDEFG", then you can get by without the RIGHT$ command without too much difficulty:

```
10 A$ = "ABCDEFG"
20 PRINT MID$(A$,LEN(A$)-2,3)
```

What is the length of the string? Size seven, that's right, so LEN(A$) is 7-2 (=5), and 3 equals 3, so the PRINT output is the three characters starting at position 5:

```
EFG        Quod Erat Demonstrandum, or words to that effect.
```

LEFT$ is just as easy if not easier. For instance if you wanted LEFT$(A$,4) then you can use MID$(A$,1,4) instead; a similar example but to simulate RIGHT$(A$,3) is to use MID$(A$,LEN(A$)-2,3)

General Simulation of RIGHT$ using MID$

This can be expressed as:-

```
RIGHT$(A$,Y) = MID$(A$,LEN(A$)-Y+1,Y)
```

In both of these expressions (i.e. the ones to the left and right of the equals sign above), the last Y characters of A$ are produced.

Proof:
Run this program:

```
10 DOFF: 1INE1: CLS              Initialise
20 PRINT "ENTER SOME STRING"     Get working string
30 A$ = INPUT$
40 FOR Y = 1 TO LEN(A$)          To produce all the right strings
50 B$ = MID$(A$,LEN(A$)-Y+1,Y)   Simulate RIGHT$(A$,Y)
60 RPINT B$
70 NEXT Y
```

This will produce all possible RIGHT$s for any given string and so prove the equivalence of the derived function to the desired function (if you'll excuse the alliteration).

Note that in practice some complication would be recommended, i.e.

```
10~30 As above
40 X = LEN(A$)                   ;X set to length A$
50 P = X+1                       ;P set to length +1
60 FOR Y = 1 TO X
70 B$ = MID$(A$,P-Y,Y)
80 PRINT B$
90 NEXT Y
```

The function in line 70 is now simpler to write, and the program should run faster, as LEN(A$) is outside the loop and is only calculated once.

Of course, if you are a really creative programmer, you will want to mix string commands in the same expression. Not being highly creative myself, I can only offer the following as an example:

```
10 A$ = INPUT$
20 FOR J = 1 TO LEN(A$)
30 PRINT MID$(A$,1,RND(LEN(A$)))
40 NEXT J
```

The last point to mention on strings is that they can be used with the normal INPUT command, (for assigning numbers to an ordinary variable),

e.g.

```
10 A$ = "NEXT"
20 X = INPUT A$ X
```

A few short and very simple programs follow to demonstrate the string syntax:

Example 1.

```
10 DOFF: CLS: PAGE
20 P."ENTER STRING
30 A$=INPUT$
40 FOR J = 1 TO LEN(A$)
50 P.MID$(A$,1,J)
60 NEXT J
70 SCROLL
80 STOP
```

It has been suggested that a use of the above program would enable a user to manufacture his own personalised wall-paper, to add that little touch of "je ne sais quoi" to his or her computer shack.

Example 2.

```
10 PAGE: CLS: DOFF
20 A$="12345678
25 B$="ABCDEFGH
30 C$=MID$(A$,1,3)+MID$(B$,1,3)
40 A=LEN(C$)
50 PRINT "A$=",A$
60 P.
70 PRINT "B$=",B$
80 P.
90 PRINT "C$=",C$
100 P.
110 PRINT "C$ IS",%1,A,"CHARACTERS LONG"
120 MID$(C$,4,3)=MID$(A$,1,3)
130 P.:PRINT "NOW C$=",C$:P.:P."PLEASE",
140 P."TYPE YES OR NO ",
150 D$=INPUT$
160 IF MID$(D$,1,1)="Y" P."YOU ACCEPT":G.180
170 P."YOU DECLINE
180 SCROLL
190 STOP
```

If you want to demonstrate the fallibility of computers, instead of answering "YES" or "NO" at line 150, answer "YOU MUST BE JOKING!"

Extract from User Group Newsletter No. 1 BITS OF BASIC.[1]

Strings and Poke.

The strings on ZYBASIC V2.03 are held in store from #8100 upwards. 26 strings exist, each starting on a 256 byte boundary.

A$ 8100	B$ 8200	C$ 8300	D$ 8400
E$ 8500	F$ 8600	G$ 8700	H$ 8800
I$ 8900	J$ 8A00	K$ 8B00	L$ 8C00
M$ 8D00	N$ 8E00	O$ 8F00	P$ 9000
Q$ 9100	R$ 9200	S$ 9300	T$ 9400
U$ 9500	V$ 9600	W$ 9700	X$ 9800
Y$ 9900	Z$ 9A00		

To see a string try the following (press carriage return after each line):

```
10 A$="AAAA"
20 Z$="ZZZZ"
RUN
BYE
T 8100
T 9A00
```

and both strings are now on the screen as thay are held in real store. An important fact about the strings is that thay are terminated by an FF character. To see that try:-

```
10 A$="AAAAAA"
20 A$="ZZ"
RUN
BYE
T 8100
```

and the FF code is in the third position. This tells ZYBASIC that is only two characters in length. The above facts mean we can do to strings what very few BASICS will allow: poke to them! The only rule to follow is to poke an FF at the end of the string. A use for all of this is in fast graphic displays. So:-

a string is 256 (-1) characters long, take A$ as an example: 8100-81FF. Given that 81FF must be set to FF than we have 255 characters with which to work. A VDU line is 32 characters long, and so a string covers 7.6 lines on the screen. If we ignore the 0.6, then 7 screen lines can be written with one PRINT line! i.e. a string could hold a monster for a graphics game, or two strings could hold the 14 line short range scan display for the game of STARTREK. Updates would be POKEd into the string, and display would be

```
PAGE:PRINT $:PRINT $.
```

Another use is for USR routines, hold them in a DATA statement, read them into the string and USR the string address to execute your code. To start you off, POKEing into strings, here is a utility to null them.

(Extract 1 continued)

```
998 ! Null String S
999 ! entry S has string address
1000 FOR Z9=0 TO 223     ;covering 7 lines
1010 POKE S+Z9,0         ;null a character
1020 NEXT Z9             ;null 224 characters
1030 POKE S+224,#FF      ;plant the terminator
1040 RETURN              ;exit string is nulled
```

Enter with the string address in S, ie take A$ at 8100

```
10 S=#8100               ;S points to A$
20 GOSUB 1000            ;clear S string,A$
30 PRINT A$              ;print the result
40 STOP                  ;end
```

For fun, change line 1010 to

```
1010 POKE S+Z9,d
```

and add

```
15 INPUT D
35 GOTO 10
```

Now RUN and give 6 in response to the data request.

Finally try this program which shows how 3 PRINT $ statments can cover the major display area on the screen:-

```
10 CLS:DOFF:PAGE:LINE1
20 S=#8100
30 FOR D=0 TO #7E
40 GOSUB 1000
50 P.A$:LINE 8
60 P.A$:LINE 15
70 P.A$:LINE 1
80 NEXT D
90 GOTO 20
1000 FOR Z=0 TO 223
1010 POKE Z9+S,D
1020 NEXT Z9
1030 POKE S+224,#FF
1040 RETURN
```

Please note: the character codes #08 and #7F are backspace and so produce a funny response when printed as a string. Also code 12 (#0C) is CTRL L which is clear screen, so you get 672 clear screens! And finally 13 is CR, so you 672 carriage returns!

With those restrictions in mind I hope you get some fun out of POKEing to a string.

BYE
Bob Eldridge

APPENDIX C

APPENDIX C

4.3 Appendix C

EXAMPLE ROUTINES IN ZYBASIC

These are for your study and amusement. They may be freely used as part of your programs.

PROGRAM 1.
An array shuffler (for cards and things). This routine takes an array and shuffles the contents.

```
100 INPUT "Enter size of array",N  ;Get size (52 perhaps)
110 !
120 FOR I = 1 TO N                 ;First fill the array with numbers
130 @(I)=I                         ;in numeric ascending order (cards)
140 NEXT I                         ;
150 !
160 GOSUB 500                      ;Print the result
170 !
180 FOR I = 1 TO N                 ;Shuffle the array (of cards)
190 X = INT(RND(0)*I)+1
200 D = @(I)
210 @(I) = @(X)
220 @(X) = D
230 NEXT I
240 !
250 GOSUB 500                      ;Print the result
260 !
270 STOP
500 ! ARRAY PRINTER
510 FOR I = 1 TO N                 ;Print the array of N locations
520 PRINT @(I),
530 NEXT I
540 RETURN
```

PROGRAM 2. MONSTER MASH.

For the benefit of those users who have a VDU-K single card VDU, with a CG02.01 character generator, here is a program called "Monster Mash". A random maze is prepared (itself a most interesting activity), four "monsters" are positioned within the maze, and the object is to "mash" the monsters with the minimum number of "bounces" on the sides of the maze.

It has been suggested that a pleasant evening may be enjoyed by playing the game, noting your score, having a glass of whisky, playing again, noting the worsening of your score, having another glass of whisky, and so on. It is surprising how interested in computers people will become when the game is played in this manner.

Apart from demonstrating some commands in the BASIC, particularly SET, RESET, and POINT, it has the following objects:

1. To show how quick ZYBASIC 2 is, in that it can control the animation of a ball, check for rebounds, keep track of a running score, and control a programmable sound generator, all in "real time". Many systems would not be able to do this without resorting to machine code.

2. To demonstrate the high quality of the VDU-K display, its extremely steady (i.e. no "snow") display, and the use of the graphics characters in the standard CG02.01 character generator.

To achieve the results mentioned above, the system has to be run with a CPU frequency of at least 4 MHz, and if the sound output is required (which greatly enhances the game) then you will need a PSG-1 sound generator card. This unfortunately only exists in prototype form at the moment, but some details can be supplied to interested users. A much better keyboard response is obtained if a latched keyboard interface is used. The new keyboard interface card LKP-1 includes the circuit to provide this.

MONSTER MASH

```
10 G.1120
20 IF J=69 E=1:G.70
30 IF J=68 E=2:G.70
40 IF J=79 E=3:G.70
50 IF J=80 E=4:G.70
60 G.80
70 OUT L,E*20
80 C=A:D=B
90 ON E G.100,110,120,130
100 B=B+1:G.140
110 B=B-1:G.140
120 A=A-1:G.140
130 A=A+1
140 M=POINT(A,B)
150 OUT L,0
160 IF M=0 G.230
170 IF M=M1 G.860
```

MONSTER MASH Listing (continued)

```
180 OUT L,24
190 C1=C1+1:OUT L,0
210 LINE 3:P."BOUNCES ",%1,C1,
220 A=C:B=D
230 RESET(C,D):SET(A,B)
240 INK.J
250 IF J<>0 G.20
260 IF M=0 G.80
270 J=RND(4):IF J=E G.270
280 E=J:G.80
290 !
300 F=300
310 X=RND(20):Y=RND(13)
320 Z=Y*21+X
330 @(F)=Z
340 FOR W=0 TO 275:@(W)=0:NEXT W
350 @(Z)=16
360 IF X=0 G.380
370 Z=Y*21+X-1:GOS.450
380 IF X=20 G.400
390 Z=Y*21+X+1:GOS.450
400 IF Y=0 G.420
410 Z=(Y-1)*21+X:GOS.450
420 IF Y=12 G.470
430 Z=(Y+1)*21+X:GOS.450
440 G.470
450 IF@(Z)<>0 RET.
460 F=F+1:@(F)=Z:@(Z)=-1:RET.
470 IF F=300 G.670
480 G=300+RND(F-300):Z=@(G)
490 E1=E1-1:LINE 15:P.%11,E1,"AND COUNTING",
500 FOR H=G TO F:@(H)=@(H+1):NEXT H
510 F=F-1
520 Y=I.(Z/21):X=Z-Y*21
530 N=N+1:IF N>4 N=1
540 ON N G.550,580,610,640
550 IF X=20 G.530
560 IF @(Z+1)<1G.530
570 @(Z)=1:@(Z+1)=@(Z+1)+4:G.360
580 IF Y=12G.530
590 IF @(Z+21)<1G.530
600 @(Z)=2:@(Z+21)=@(Z+21)+8:G.360
610 IF X=0G.530
620 IF @(Z-1)<1G.530
630 @(Z)=4:@(Z-1)=@(Z-1)+1:G.360
640 IF Y=0G.530
650 IF @(Z-21)<1G.530
660 @(Z)=8:@(Z-21)=@(Z-21)+2:G.360
670 !
680 GOS.820
690 FOR P=0 TO 12
700 V=41-P*3:U=-3
710 FOR S=0 TO 20
```

MONSTER MASH Listing (continued)

```
 720 U=U+3
 730 R=@(P*21+S)
 740 IF R>15 R=R-16
 750 IF R<8 R=R+8:SET(U,V):SET(U+1,V):SET(U+2,V):SET(U+3,V)
 760 IF R<12 SET(U,V):SET(U,V-1):SET(U,V-2):SET(U,V-3)
 770 NEXT S
 780 SET(63,V):SET(63,V-1):SET(63,V-2):SET(63,V-3)
 790 NEXT P
 800 FOR S=0 TO 63:SET(S,2):NEXT S
 810 RET.
 820 LINE 1
 830 CLS
 840 P."           MONSTER MASH
 850 RET.
 860 D1=D1+1
 870 OUT L1,0
 880 IF D1<>4 G.230
 890 RESET(A,B)
 900 FOR J=1 TO 8000:NEXT J
 910 GOS.820:P.:P.
 930 P."BOUNCE COUNT ",%1,C1
 940 IF A1=0 A1=C1:G.960
 950 IF C1>A1 P.:G.980
 960 P."WHICH IS THE LOWEST SO FAR.
 970 A1=C1:P.
 980 P.:P.:P."YOU CAN NOW CHOOSE TO:-
 990 P.:P." A..PLAY THE SAME MAZE AGAIN.
1000 P.:P."OR
1010 P.:P." B..TRY YOUR SKILL ON A NEW ONE.
1020 P." NOTE: CHOICE A WILL PRODUCE
1022 P."       THE MAP INSTANTLY AS
1024 P."       IT IS ALREADY DRAWN.
1026 P.
1030 P."PRESS A OR B TO INDICATE CHOICE
1040 INK.J
1050 IF J=#42:GOS.820:G.1190
1060 IF J<>#41 G.1040
1070 GOS.820
1080 LINE 2
1090 P."BEST SCORE ",%1,A1
1100 GOS.690
1110 G.1300
1120 PAGE:DOFF:GOS.820
1130 P.:INK.J
1140 P."IF YOU WOULD LIKE INSTRUCTIONS"
1150 P."PRESS Y, OTHERWISE PRESS N
1160 INK.J
1170 IF J=#59 G.1690
1180 IF J<>#4EG.1160
1190 N=1:P.
1200 P.
1210 P."I REQUIRE A SHORT TIME TO SCAN
1220 P."THE MAZE AND PRODUCE A PLAN.
```

MONSTER MASH Listing (continued)

```
1230 P."PLEASE WAIT A WHILE.
1240 P.
1250 P."THE GAME WILL BEGIN WHEN THE
1260 P."COUNTDOWN IS AT ZERO
1270 A1=0
1280 E1=273
1290 GOS.300
1300 A=1:B=3
1310 SET(A,B)
1320 M1=#1D:C1=0:D1=0
1330 POKE #F0A2,M1
1340 POKE #F16E,M1
1350 POKE #F0BD,M1
1360 POKE #F29D,M1
1370 RESTORE:GOS.1550
1380 X2=#F2E0
1390 FOR Y2=1 TO 29
1400 READ Z2
1410 POKE X2+Y2,Z2
1420 NEXT Y2
1430 RESET(A,B)
1440 FOR J=0 TO 200:NEXT J
1450 INK.J
1460 SET(A,B)
1470 IF J<>0 G.20
1480 FOR J=0 TO 200:NEXT J
1490 G.1430
1500 DATA #45,#2D,#55,#50,#20
1510 DATA #20,#44,#2D,#44,#4F
1520 DATA #57,#4E,#20,#20,#4F
1530 DATA #2D,#4C,#45,#46,#54,#20,#20
1540 DATA #50,#2D,#52,#49,#47,#48,#54
1550 K=#C0:L=#C1:K1=#C2:L1=#C3
1560 OUT K,7:OUT L,#FE
1570 OUT K,8:OUT L,3
1580 OUT K,0
1590 OUT K1,6:OUT L1,31
1600 OUT K1,7:OUT L1,7
1610 OUT K1,8:OUT L1,16
1620 OUT K1,9:OUT L1,16
1630 OUT K1,10:OUT L1,16
1640 OUT K1,12:OUT L1,20
1650 OUT K1,13
1660 RET.
1670 FOR J=0 TO 6000:NEXT J
1680 RET.
1690 N=1:GOS.820
1700 P."IN A MAZE IS A BALL,o.
1710 P.:GOS.1670
1720 P."YOU CAN ALTER THE DIRECTION OF
1730 P."THIS BALL AND YOU DO SO WITH
1740 P."THE IDEA OF ZAPPING THE FOUR
1750 P."MONSTERS THAT ARE IN THE MAZE.
```

MONSTER MASH Listing (concluded)

```
1760 P.:GOS.1670
1770 P."WALL BOUNCES ARE FAULTS AND A
1780 P."RANDOM REBOUND WILL RESULT.
1790 GOS.1670:P.:P."ALSO A BOUNCE COUNT IS KEPT
1800 P."SO YOU CAN ASSESS YOUR SKILL
1810 P." (LESS BOUNCES MORE SKILL).
1820 REM KEYS SET @20,30,40,50
1830 P.
1840 GOS.1670
1850 P."THE CONTROL KEYS ARE:-
1860 P.:P."            E-up
1870 P.:P."    O-left      P-right
1880 P.:P."          D-down
1890 P.
1900 GOS.1670
1910 REM DISCARD STRAY KEYS
1920 INK.J
1930 P."PRESS ANY KEY TO START.
1940 INK.J
1950 IF J=0G.1940
1960 GOS.820:N=1:G.1190
1970 REM
1980 REM PROGRAM AVAILIABLE TO ZYBASIC USERS.
1990 REM COPYRIGHT R.E. 1981
2000 REM ----------------------
```

Line 1700 mentions a ball, o. This in fact is a "pixel" graphics character which is obtained by entering CTRL-A into the PRINT statement.

Programming Hint: (The method used to gain speed in Monster Mash)

Whenever a GOTO or GOSUB is encountered ZYBASIC goes right back to the first line in the program and searches line at a time until it finds a line number which matches the destination of the GOTO or GOSUB in question. Therefore subroutines placed at the beginning of a program will be found and executed much more quickly than ones placed towards the end. If speed of access to a subroutine or program branch is important then place that part of the program at the beginning of the program. At the very beginning a GOTO will have to be placed to jump over all the subroutines at the beginning. e.g.

```
10 GOTO 2000              Unconditional jump to main program
     SUB 1                First Subroutine
     SUB 2                Second Subroutine
     SUB 3                Third Subroutine
   BRANCH HERE LOOP BACK
1999 STOP                 Protective stop
2000 Main Program
```

Extract from User Group Newsletter No.1 BITS OF BASIC.[1]

Simulating Arrays.

ZYBASIC has only one array. It is single dimensioned and is accessed by:-

```
@ ( some number )
```

The "some number" can also be an expression or a variable name.

So these are legal :-

```
@ (3+4)
@  (A)
@ (A+B)
```

If I need two single dimensioned arrays I can simulate them by using displacement indexing :-

```
10 A=0
20 B=100
30 @(A+3)=6
40 @(B+3)=6
```

Here the third item in each array has been set to the value 6.

To be even more flexible I should specify my index as a variable. So using Y as my index gives :-

```
10 A=0
20 B=100
30 Y=3
40 GOSUB 1000
99 STOP

1000 X=1
1010 IF @(A+Y) <> @(B+Y)  X=0
1020 RETURN
```

In the subroutine flag X has been set to zero if both items indexed by Y in the arrays A and B are not equal.

To take this idea one stage further suppose that you need a two dimensioned array of size 5,4:

```
     :  1  :  2  :  3  :  4  :  5  :  :
     :-----:-----:-----:-----:-----:  :
   1:  1  :  2  :  3  :  4  :  5  :  :
     :-----:-----:-----:-----:-----:  :
   2:  6  :  7  :  8  :  9  : 10  :  :
     :-----:-----:-----:-----:-----:  N
   3: 11  : 12  : 13  : 14* : 15  :  :
     :-----:-----:-----:-----:-----:  :
   4: 16  : 15  : 18  : 19  : 20  :  :
     :-----:-----:-----:-----:-----:  :
     ---------------X---------------
     ---------------P---------------
```

To access position 14 using the horizontal and vertical indices will require an equation to convert both indices to the real position:-

Let N be the vertical index required
Let P be the horizontal index required
Let X be the horizontal dimension of the array

So in our case
X = 5
N = 3
these index item 14
P = 4

Using the expression N x X + P - X will produce the real index.

So in an example program:

```
10 X=5
20 N=3
30 P=4
40 GOSUB 1000
90 STOP

1000 A=@(N*X+P-X)
1010 RETURN
```

Now A has the value of the two dimensioned array held at index position 4,3.

More than one array can be formed by using the displacement method previously described so :-

```
10 A=0
20 B=100
    :
90 GOSUB 1000
    :
    :
1000 @(B+(N*X+P-X))=Y
1010 RETURN
```

Here the value Y has been placed in the two dimensioned array B indexed by P,N.

To prove the above RUN the following program, which will print a two dimensioned array as indices and real positions.

```
10 X=5
20 P."  !  1  2  3  4  5"
30 P."--!---------------"
40 FOR N=1 TO 4
50 P.%1,N,"!",
60 FOR P=1 TO X
70 P.%2,N*X+P-X,
80 NEXT P
90 P.
100 NEXT N
```

Finally can anyone produce the expression for three dimensioned arrays?

Solutions to the User Group please.

BYE
Bob Eldridge

(Extract from Interaktion Newsletter No. 1 reprinted with kind permission of Interaktion User Group)

EXTRACT FROM USER GROUP NEWSLETTER No. 2 More from Your ZYBASIC

More from Your ZYBASIC.

A number of members have commented on the lack of arithmetic and trigonometrical functions in ZYBASIC. This prompted me to look for some way of providing at least the most common functions. I have therefore listed below four Subroutines, which provide the user with:-

Square Roots	(SQR(X)) *(see note at foot of page)
Exponentiation	(X to the Power Y)
Logarithms	(Log X base e & base 10)
Exponential	(EXP (X))

All except the Exponentiation routine may be used independently but the Exponentiation routine uses both the LOG & EXP functions.

Square Root.

Gives: SQR (X) = Y (uses W & Z internally)

```
2990 S.
3000 ! SQR (X) = Y
3010 IF X= 0 Y=0:RET.
3020 IF X>0 G.3040
3030 P."Root of NEG Number!":S.
3040 Y=X*0.5:Z=0
3050 W=(X/Y-Y)*0.5
3060 IF (W=0)+(W=Z) RET.
3070 Y=Y+W:Z=W:G.3050
```

When I entered X = 198.32 out popped 14.0826; my Commodore calculator says 14.082613 (not bad!).

Exponentiation.

Gives: X↑Y = P (If X is less than zero, Y must be an odd integer. The routine uses E,L,A,B,C internally. The value of X is changed. LOG & EXP routines are required.)

```
3100 ! X↑Y =P
3110 P=1:E=0:IF Y = 0 RET.
3120 IF (X<0)*(INT(Y)=Y) P=1-2*Y+4*INT(Y/2):X=-X
3130 IF X<>0 GOS.3200:X=Y*L:GOS.3300
3140 P=P*E:RET.
```

Note line 3130 calls the LOG & EXP routines.

*I have just found out that ZYBASIC already has square roots!

Logarithms (Natural & Common).

Gives: LOG (X) (base e)=L
LOG (X) (base 10)=X
uses A,B internally. Value of X is changed.

```
3190 ! LOG(X) base e = L LOG(X) base 10 = X
3200 E=0:IF X<0 P."LOG undefined at ",%1,X:S.
3210 A=1:C=0.5
3220 IF X>A X=C*X: E=E+A: G.3220
3230 IF X<C X=X+X: E=E-A: G.3230
3240 X=(X-0.707107)/(X+0.707107): L=X*X
3250 L=(((0.598979*L+0.961471)*L+2.88539)*X+E-0.5)*0.693147
3260 IF ABS(L)<1E-6   L=0
3270 X=L*0.4342945:RET
```

Using X=4 in the above produced L=1.38629 & X=0.60206 compared with 1.3862943 & 0.6020599 from the calculator.

Exponential.

Gives: EXP(X) = E (uses L & A internally. Value of X changed)

```
3290 ! EXP(X) = E
3300 L=INT(1.4427*X)+1:IF L<127 G.3330
3310 IF X>0 P."Overflow ":S.
3320 E = 0 :RET.
3330 E=0.693147*L-X:A=1.32988E-2-1.41316E-4*E
3340 E=(((A-0.166665)*E+0.5)*E-1)*E+1:A=2
3350 IF L<0 A=0.5: L=L-1:IF L=0 RET.
3360 F.X=1 TO L: E=A*E: N.X: RET.
```

I have tried out all the above and they all seem to work OK but if you want to fly your spaceship by them don't ask me to come along.

Next issue I hope to have ArcSine, ArcCosine and ArcTangent. By the way Tangent is easily found as:

$$TAN(X) = \frac{SIN(X)}{COS(X)}$$

If you have any special function routines please let me have them some user could be stuck in space trying to get home all for the lack of your program.

Pete Vella

(Extract from Interaktion Newsletter No. 2 reprinted with kind permission of Interaktion User Group)

EXTRACT FROM USER GROUP NEWSLETTER Nos. 3 & 4 More from Your ZYBASIC

More from Your ZYBASIC.

ArcSine. Gives:

ArcSine(S), angle whose sine is S, where 0<=S<<1 (uses X,Y internally):

```
3400  X=S: IF ABS ( S)<= 0.707107 G.3450
3410  X=1-S*S:IFX<0 P. S;"IS OUT OF RANGE":S.
3420  W=X/2:Z=0  3430  Y=(X/W-W)/2 : IF (Y=0)+
      (Y=Z) X=W : G. 3450  3440  W=W+Y:Z=Y :G.
      3430
3450  Y=X+X*X*X/6+X*X*X*X*X*0.075+X*X*X*X*X*X*
      X*4.464286E-2
3460  W=Y+X*X*X*X*X*X*X*X*X*3.038194E-2
3470  IF ABS(S)> 0.707107 W=1.570796-W
3480  Y=W*57.29578:R.
```

When run this routine will give ArcSine (S) in degrees as Y and in radians as W.

ArcCosine.

Gives ArcCos(S), angle whose cosine is S, where 0<=S<=1 (uses X,Z internally) and uses the ArcSine subroutine:

```
3500 GOS.3400:Y=90-Y:W=1.570796-W:R.
```

When run gives ArcCos (S) in degrees as Y and in radians as W.

ArcTangent.

Gives ArcTan(S), angle whose tangent is X.

(Uses B,T internally and the value of X is changed); uses the Sign subroutine on the next page.

```
3600  GOS.3700:X=ABS(X):C=0
3610  IF X>1 C=1:X=1/X
3620  A=X*X
3630  B=((2.86623E-3*A-1.61657E-2)*A+4.29096E-2)*A
3640  B=((((B-7.5289E-2)*A+0.106563)*A-0.142089)
      *A+0.199936)*A
3650  A=((B-0.333332)*A+1)*X
3660  IF C=1 A=1.570796-A
3670  A=T*A:C=A*57.29578:R.
```

When run gives ArcTan (S) in degrees as C and in radians as A.

Extract from Newsletters 3 & 4 (continued)

Sign. (Used with the previous routines).

Gives SGN(X), the sign-component of X.

```
3700 IF X<0 T=-1
3710 IF X=0 T=0
3720 IF X>0 T=1
3730 R.
```

When run T is set to -1 if X is negative, T set to 0 if X = 0 and T set to 1 if X is positive.

Pete Vella

GRAPHICAL PLOTTER.

This program sent in by M. Cottrell for use with ZYBASIC 2 allows the plotting of graphical functions.Instructions are written in to the program and are fairly obvious to use. Briefly, a blank graph is displayed on which can be included:

1. X & Y axis information.

2. Plots of whatever function is inserted into line 170, (i.e. at present Y=SINE (X)*A).

3. Individual plots to be linked up by the computer. The plots of functions can be overlaid on top of the linked plots - the linked plots put first. Although the number of pixels is limited it will display a SINE Wave using data V1=20, X=0 , X1=10 , A=20 , B C + D = 0 . The graph can be redrawn without disturbing AXIS information already on display.

```
1 CLS:LINE1
2 P."GRAPH PLOTTING INSTRUCTIONS": P."~~~~~~~~~~~~~~~~~~~~~~~~~~~~":
  P."40 VERTICAL  PLOTS AVAILABLE":
  P."56 HORIZONTAL PLOTS AVAILABLE":P."INPUT DATA V1(0 TO 40)
  SHIFTS ":P."THE HORIZONTAL AXIS UPWARDS.":."SCALING IS
  ACCORDING TO INPUT ":P."VARIABLES.
3 P."X IS INDEPENDENT VARIABLE":P."STARTING VALUE.":
  P."X1 IS INCREMENTAL CHANGE OF X.":
  P."TO ENTER YOUR GRAPHIC FUNCTION":
  P."KEY IN 1 THEN 170 Y=(formula) ":
  P."USING VARIABLES X,A,B,C,D":P."AND RERUN PROGRAM.
4 P."PLOTS MAY ALSO BE ENTERED WITH":
  P."NO FORMULA WHICH WILL BE DRAWN":P."BY THE COMPUTER.":
  P."SELECT FOR DATA,GRAPH,AXIS -":
  P."SCALE OR THESE INSTRUCTIONS BY":P."KEYING IN C...."
7 INK.A:IF  A=#43 GOTO 10
```

GRAPHICAL PLOTTER. (continued)

```
8 IF A=#31 GOTO 280
9 IF A=0 GOTO 7
10 DOFF:CLS:LINE1:GOTO 15
12 P."                                ":
   FOR A1 =4 TO 60:FOR A2 =6 TO 47:RESET(A1,A2):NEXT A2:NEXT A1
15 POKE#F002,#13:POKE#F01E,#12:POKE#F282,#11:POKE#F29E,#10
38 A=0
40 J=#F003:POKE(J+A),#19:A=A+2:IF A<=26GOTO 40
48 A=0
50 J=#F283:POKE(A+J),#19:A=A+2:IF A<=26 GOTO 50
52 A=0
55 J=#F004:POKE (A+J),#17:A=A+2:IF A<=25 GOTO 55
57 A=0
60 J=#F284:POKE(A+J),#16:A=A+2:IF A<=25 GOTO 60
68 A=0
70 J=#F022:POKE(A+J),#18:A=A+28:POKE(A+J),#18:A=A+36
72 IF A<600GOTO70
73 A=0
80 J=#F042:POKE(A+J),#15:A=A+64:IF A<550 GOTO 80
82 A=0
90 J=#F05E:POKE(A+J),#14:A=A+64:IF A<550 GOTO 90
92 A=0
95 FOR A=1 TO 26:J=#F002:J=J+A:IF PEEK(J)=#17 GOTO 100
97 GOTO 105
100 FOR B=1 TO 19: POKE(J+32*B),#18:NEXT B
105 NEXT A
107 J=#F002
110 J=J+64:IF J>#F269 GOTO 180
115 A=0
120 A=A+1:POKE(A+J),#19:A=A+1:IF A>27 GOTO  110
122 POKE(A+J),#1A:GOTO 120
123 P."                                ":LINE 23:GOTO 180
126 P."                                ":J=#F2A2:
    FOR A=1TO29:POKE(J),04
127 INK.X:IF X=0 GOTO 127
128 POKE(J),X:J=J+1:NEXT A
129 J=#F280
131 GOSUB 140
132 IF J<=#F002 GOTO 165
133 J=J-66:GOTO 131
140 FOR A=1TO2:POKE(J),04
142 INK.Y:IF Y=0 GOTO 142
144 POKE(J),Y:J=J+1:NEXTA
146 RETURN
165 GOTO 180
168 P.:INPUT"VERTICAL SHIFT(0-40)",V1:
    P.:INPUT"INDEPENDENT VARIABLE",X:
    P.:INPUT"CHANGE IN X PER PLOT",X1:
    P.:INPUT"VARIABLE",A:P.:INPUT"VARIABLE",B:
    P.:INPUT"VARIABLE",C:P.:INPUT"VARIABLE",D:P.:H=1
170 Y =A*SIN(X)
171 Y=Y+V1
172 IF Y+6>ABS(46) GOTO 180
```

GRAPHICAL PLOTTER. (concluded)

```
173 IF H+4>ABS(59) GOTO 180
174 IF Y+6<ABS(6) GOTO 180
175 SET(H+4,Y+7):H=H+1:X=X+X1:GOTO 170
180 LINE 23:P."KEY.Data,Graph,Axis,Plot,Instr"
185 INK.A1:IF A1=0 GOTO 185
186 IF A1=#47 GOTO 12
188 IF A1=#44 GOTO 168
190 IF A1 =#41 GOTO 126
191 IF A1=#50 GOTO 200
195 IF A1=#49 GOTO 1
198 GOTO180
200 INPUT"NUMBER OF PLOTS?",P:FOR Q=1TO P:P."PLOT NUMBER",Q:
    INPUT"CO-ORDINATE",X:P."PLOT NUMBER",Q:
    INPUT"CO-ORDINATE ",Y:SET(X+4,Y+7):NEXT Q
205 P."SEARCHING FOR PLOT"
210 Z=0
214 FOR X=4 TO 60:FOR Y=7 TO 47:IF POINT(X,Y) =-1 GOTO 220
216 GOTO 250
220 IF Z=0 GOTO 240
222 S=(Y-Y1)/(X-X1)
230 Y2=Y1+S*X2:SET(X1+X2,Y2):X2=X2+1:IF X1+X2=X GOTO 240
235 GOTO230
240 Y1=Y:X1=X:Z=Z+1:X2=0
242 IF ABS(Y)> 47 GOTO 180
244 IF ABS(Y)< 7 GOTO 180
246 IF X> 60 GOTO 180
248 IF Z=P GOTO 180
250 NEXTY:NEXTX
260 GOTO 180
280 STOP
```

(Extract from Interaktion Newsletters 3 & 4 reprinted with kind permission of Interaktion User Group)

4.4 Appendix D

NOTES ON OLDER "KEMITRON" SYSTEMS

Although the early "Kemitron" system from which Interak 1 is derived is now several years old it is not so different as to cause any great problems. Some points to bear in mind are as follows:

1. Kemitron VDU-A, B, G. This three card set has largely been superceded by the Interak VDU-K. (The similarity is no coincidence, the VDU-K was designed to be a compatible single card replacement.) Features of the VDU-K which the Kemitron 3-card set did not have, and which have been used by ZYBASIC are:

   (i) Upper and lower case letters. Lower case letters are not available on the Kemitron A,B,G and for example the sign-on message will appear largely as gibberish if the VDU-K is not used.

   (ii) Block Graphics Characters. These are not available on the Kemitron VDU and so SET, RESET and POINT commands have little or no use without the VDU-K.

   (iii) Anti-Snow Feature. This is only available on the VDU-K, but as it is only a cosmetic feature it will not affect the running of the programs. The "snow" only becomes apparent in fast-moving screen displays, which are already largely denied to Kemitron System users because of (ii) above.

2. Kemitron DCR-6. The Interak 1 equivalent board is the Latched Keyboard Port type LKP-1. The benefit the LKP-1 offers is that it uses a latch to store a key code until it can be read by the operating program. The DCR-6 has no such latch and if a key is pressed when the program running is otherwise engaged, that key code is lost. The effect that a ZYBASIC user will notice that it will be much harder to use the CTRL-S to interrupt a running program if the DCR-6 board is used: repeated tapping of the key will be needed, as it is only a matter of luck for a read of the keyboard to be taking place when a key is being pressed. Luck does not enter into it when the LKP-1 card is used - just one key press is enough.

   For similar reasons only the LKP-1 will give reliable results when INKEY is used; the use of the INKEY command will be quite limited with the DCR-6 board.

3. Kemitron TPA-2 Tape Interface. This is limited by its design to a maximum data rate of 300 baud. The Interak 1 DTI-1 card allows rates up to 2400 baud, but there is no reason why the TPA-2 cannot be retained if 300 baud is adequate for the user's purpose.

APPENDIX E APPENDIX E

4.5 Appendix E

UPDATES ON VERSIONS OF ZYBASIC 2

The cost and time involved in producing a manual such as this are such that it is only economic to revise it at relatively infrequent intervals. The Interaktion User Group Newsletter is published a number of times each year and so that is the place where updates and the like can be given.

However at the time this manual was written plans were being made to make a minor alteration to the previous version (2.03) to make it 2.04 and so this change can be documented here. The EPROM based version may remain at 2.03, because it is not so convenient to change as the tape version. The bytes in version 2.03 which can be changed to make it into version 2.04 are detailed below. (It is very easy for the user to change the tape based version - the procedure is to use the monitor program to load ZYBASIC, make the modification, and save the new version.)

Address (V2.03A)	Address (V2.03C)	From	To
A070	C070	33	34
A6B9	C6B9	CD07AA	000000

After the change is made a space will no longer be inserted automatically by the PRINT command when it is used to print a string.

e.g.

10 PRINT "BILL","BEN" will now print BILLBEN. Before the change this line would have printed BILL BEN (i.e. with an automatic space between the two words). It is clearly not a matter of earth shattering importance but the alteration was introduced to suit a customer who found the automatic space inconvenient. In version 2.04 a space can be introduced if need be, by rewriting the line with an extra comma:
i.e.

10 PRINT "BILL",,"BEN"

Another User Option:

To help the user locate the cursor on the screen (particularly in PAGE mode), a line of dots is added by ZYBASIC on the line which contains the cursor. The default is that the dots are on, and the command DOFF must be used to turn them off (and DON to turn them on again). If you would prefer the ZYBASIC default to be dots off you can change the following byte. (The byte to change in the EPROM version is given as well, but it is of course much easier to change in the RAM-based tape version):

Address (V2.03A)	Address (V2.03C)	From	To
A030	C030	2E	00

C O N C L U S I O N

THE END .....OR IS IT THE BEGINNING! GOOD LUCK AND HAVE SOME FUN WITH THE ZYBASIC INTERPRETER (THAT'S THE POINT REMEMBER).

BYE
Bob and Dave

ZYBASIC Manual written by Bob Eldridge, with contributions from David Parkins of Greenbank Electronics, and Pete Vella of the Interaktion User Group.

Interak Computer System Software.

Published by Greenbank Electronics 1983

(This page intentionally left blank)